

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

**YÜKSEK BAŞARIMLI VE MERKEZİYETSİZ HESAPLAMA
SİSTEMLERİNDE POLİTİKA TABANLI YÖNETİM**

DOKTORA TEZİ

Taner DURSUN

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

NİSAN 2023

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

**YÜKSEK BAŞARIMLI VE MERKEZİYETSİZ HESAPLAMA
SİSTEMLERİNDE POLİTİKA TABANLI YÖNETİM**

DOKTORA TEZİ

**Taner DURSUN
(504982301)**

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Burak Berk ÜSTÜNDAĞ

NİSAN 2023

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

**POLICY BASED MANAGEMENT IN HIGH PERFORMANCE AND
DECENTRALIZED COMPUTING SYSTEMS**



Ph.D. THESIS

**Taner DURSUN
(504982301)**

Department of Computer Engineering

Computer Engineering Programme

Thesis Advisor: Prof. Dr. Burak Berk ÜSTÜNDAĞ

APRIL 2023

İTÜ, Lisansüstü Eğitim Enstitüsü'nün 504982301 numaralı Doktora Öğrencisi Taner DURSUN, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı “YÜKSEK BAŞARIMLI VE MERKEZİYETSİZ HESAPLAMA SİSTEMLERİNDE POLİTİKA TABANLI YÖNETİM” başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı : **Prof. Dr. Burak Berk ÜSTÜNDAĞ**
İstanbul Teknik Üniversitesi

Jüri Üyeleri : **Prof. Dr. Kemal BIÇAKÇI**
İstanbul Teknik Üniversitesi

Dr. Öğr. Üyesi M. Tahir SANDIKKAYA
İstanbul Teknik Üniversitesi

Prof. Dr. Osman Nuri UÇAN
Altınbaş Üniversitesi

Prof. Dr. Yıldırım YALMAN
Piri Reis Üniversitesi

Teslim Tarihi : 20 Mayıs 2022
Savunma Tarihi : 18 Nisan 2023





Değerli Hocam Prof. Dr. Ahmet Coşkun Sönmez'in anısına,



ÖNSÖZ

Danışmanım Sayın Prof. Burak Berk ÜSTÜNDAĞ'a ne kadar teşekkür etsem azdır. Bana olan güveni ve Merkeziyetsiz Teknolojiler konusundaki kapsam genişletmesi olmasaydı bu çalışma bu kadar verimle tamamlanamazdı. Ömür boyu minnettar olacağım. Sayın Jüri üyelerim, Sayın Prof. Dr. Kemal BIÇAKÇI, Prof. Dr. Osman Nuri ATEŞ, Prof. Dr. Yıldray YALMAN ve Dr. M. Tahir SANDIKKAYA'ya savunma sınavı davetimizi kabul ettikleri ve değerlendirmeye katkıları için teşekkür ederim.

Çalışmamı tamamlamam için beni cesaretlendiren Sayın Prof. Dr. Ertuğrul KARAÇUHA'ya ve her konuda yardımlarını esirgemeyen Lisanüstü Eğitim Enstitüsü'nün değerli personeline teşekkürü bir borç bilirim. Ayrıca, tezimi Blokzincir alanına uyarlamamda destekleri nedeniyle Öznur ARABACI KALKAR, Nevzat ÖZCANDAN ve BİLGEM Blokzincir Araştırma Laboratuvarı'nın diğer üyelerine de teşekkür ederim.

Makale ve bildirileri yayınladığımız dergi ve konferansların hakemlerine, değerli geri bildirimleri için teşekkür etmek istiyorum.

Birlikte çalışmak için samimi bir ortam oluşturan BİLGEM Elektronik Anahtar Yönetim Grubu üyelerine özel olarak teşekkür etmek istiyorum. Tez yazımında bana yardımcı olan Erdem BİLGİLİ, Bülent ÇAVUŞOĞLU, Bahattin DAĞ, Hasan GENÇ, Hasan ER ve Ahmet BAŞGÖZE'ye teşekkür ederim.

Çalışmam boyunca inanılmaz sabrı ve desteği için eşime teşekkür etmek istiyorum. Sürekli varlığı olmasaydı, bu tez olmazdı. Doktora yapmam için yıllardır bana sürekli destek olan aileme minnettarım ve bu tezi onlara da ithaf etmek istiyorum. Bana bu işi üstlenme cesareti verdiler.

Nisan 2022

Taner DURSUN
(Bilgisayar Mühendisi)



İÇİNDEKİLER

Sayfa

ÖNSÖZ.....	ix
İÇİNDEKİLER	xi
KISALTMALAR.....	xv
ÇİZELGE LİSTESİ.....	xvii
ŞEKİL LİSTESİ.....	xix
ÖZET.....	xxiii
SUMMARY	xxv
1. GİRİŞ	1
1.1 Tezin Amacı	2
1.2 Tezin Katkıları.....	3
1.3 Tezin Organizasyonu.....	4
2. YÜKSEK BAŞARIMLI HESAPLAMA SİSTEMLERİ	5
2.1 PDCA Mimari Sınıflaması	5
2.1.1 HTC ve HPC kavramları.....	7
2.1.2 İş, görev, RMS ve JMS kavramları.....	7
2.1.2.1 JMS fonksiyonları	9
2.1.2.2 JMS tarafından sağlanması gereken özellikler.....	10
2.1.2.3 JMS ve RMS çalışan sistemlerin sınıflandırılması	11
2.2 Küme (Cluster) Yönetim Sistemleri.....	12
2.3 Gridler	17
2.4 Bulut Bilişim Mimarileri	24
2.5 Diğer Hesaplama Mimarileri.....	25
2.6 Merkeziyetsiz Sistemler	26
2.6.1 Blokszincir mimarisi ve yapıtaşları	27
2.6.1.1 Kayıtlar, işlemler ve bloklar.....	28
2.6.1.2 Blokszincir sisteminin aktörleri.....	29
2.6.1.3 Kriptografik yapıtaşları	29
2.6.1.4 Mutabakat protokolleri.....	30
2.6.1.5 Akıllı kontratlar.....	30
2.6.2 Blokszincirin işleyişi	31
2.6.3 Blokszincir platformları.....	32
3. POLİTİKA TABANLI YÖNETİM VE POLİCE PBM ALTYAPISI	33
3.1 Politika Tabanlı Yönetim (PBM)	33
3.1.1 Genel PBM mimarisi	34
3.1.2 PBM ürünleri için gereksinimler.....	35
3.1.3 Politikalar ve tanımlama dilleri.....	35
3.1.3.1 Mantık-tabanlı diller.....	36
3.1.3.2 Kural-tabanlı diller	37
3.1.3.3 Özel tanımlama diller	37
3.1.4 PBM ürünleri.....	39

3.2 POLICE PBM Yazılımı.....	41
3.2.1 Mimari tanıtımı	42
3.2.2 POLICE politika tanımlama dili	43
3.2.2.1 Dil söz dizimi	44
3.2.2.2 Kısıt ifadeleri.....	46
3.2.2.3 Olay ifadeleri.....	46
3.2.3 POLICE politika uygulama bileşenleri	46
3.2.3.1 Etki alanı servisi	47
3.2.3.2 Politika uygulama noktası (PPEP)	48
3.2.3.3 Olay servisi (ES)	49
3.2.3.4 Yönetim arayüzü (MI).....	52
3.2.4 POLICE çatısının gerçekleştirilmesi	53
3.2.5 POLICE çıkışma kotarma yöntemi.....	54
4. HESAPLAMA SİSTEMLERİNİN PBM İLE YÖNETİLMESİ.....	59
4.1 YBH Yönetimi İçin PBM Kullanılması	59
4.2 Literatürde YBH Sistemlerinin Politika Tabanlı Yönetimi Çalışmaları	62
4.2.1 Grid ve kümelerin PBM ile yönetimi.....	64
4.2.2 Bulut sistemlerinin PBM ile yönetimi.....	85
4.2.3 Merkeziyetsiz sistemlerde PBM ile yönetim	87
4.3 Evrensel YBH Modeli	88
4.4 Police Modeli ile YBH Yönetimi	95
4.4.1 YBH politika türleri	97
4.4.2 Politikaların kısıt ifadeleri	98
5. HETEROSİM HETEROJEN SİMÜLASYON MODELİ.....	101
5.1 Giriş	101
5.2 Simülasyon Araçları	103
5.2.1 Literatürde simülasyon araçları.....	104
5.2.2 YBH amaçlı simülasyon araçları literatür incelemesi.....	106
5.3 HeteroSim.....	112
5.3.1 HeteroSim mimarisi	116
5.3.2 Simülasyon saati sorunu.....	119
5.3.3 HeteroSim gerçekleştirilmesi	120
5.3.4 HeteroSim'in PBM simülasyonlarında kullanılması.....	126
5.3.4.1 HeteroSim doğruluk analizi	129
5.3.4.2 HeteroSim ölçeklenebilirlik ve performans analizi.....	147
6. KAVRAM İSPATI ÇALIŞMASI-1: TARBİL İÇİN PBM.....	157
6.1 TARBİL Sistemin Tanıtımı	157
6.2 Analiz Çalışması-1.1 : TARBİL Sisteminin Politikalarla Yönetilebilir YBH Yapısına Dönüştürülmesi	159
6.2.1 Analiz çalışması-1.1 yol haritası	160
6.2.2 OpenStack bulut platformu	161
6.2.2.1 OpenStack servisleri.....	162
6.2.2.2 IBM bulut desteği.....	165
6.2.2.3 OpenStack platformunun politika tabanlı yönetimi	167
6.2.3 POLICE ile OpenStack yönetimi entegrasyonu.....	177
6.3 Analiz Çalışması-1.2: TARBİL Sistemi MSVD Uygulamasının Politikalarla Yönetilebilir Şekilde Paralleştirilmesi.....	180
6.3.1 TARBİL süreç yapısı ve iş yükü tipi sınıflandırması.....	181
6.3.1.1 Yük tiplerinin karakteristiğinin modellenmesi.....	184
6.3.1.2 TARBİL sistemi için YBH aracı seçimi	191

6.3.1.3 JMS araçlarının güncel durum araştırması.....	191
6.3.1.4 Seçilen JMS aracı ve mimari tasarımı.....	191
7. KAVRAM İSPATI ÇALIŞMASI-2: MERKEZİYETSİZ SİSTEMLERİN POLİTİKA TABANLI YÖNETİŞİMİ	197
7.1 Giriş	197
7.2 Benzer Çalışmalar	203
7.3 Arka Plan.....	208
7.3.1 PBM'in blokzincir yönetiřimi için kullanılabilirlięi.....	208
7.3.2 DID yönetim teknolojisi ve modelimizde kullanılma gerekçesi.....	210
7.4 Önerilen Model.....	213
7.4.1 Genel bakış.....	213
7.4.2 Mimari modeli.....	214
7.4.3 Önerilerin uygulamaya sokulması	216
7.4.4 DID tabanlı oylama řeması	218
7.4.4.1 DID mimarisi	218
7.4.4.2 Aktörlerin sisteme kaydı	218
7.4.4.3 Oylama süreçleri	219
7.4.4.4 Oy verisinin detayları.....	222
7.5 Gerçekleme ve Deneysel Sonuçlar	224
7.6 Saldırgan Tehdit Modeli.....	226
7.7 Analiz Çalışması -2.1: DAO Yönetiřimi İçin Kavramsal Çözüm Önerisi.....	230
7.7.1 DAO yönetiřim zorlukları.....	230
7.7.2 İlgili çalışmalar	234
7.7.3 DAO'lar için bir çözüm yaklaşımı.....	237
7.7.4 Sonuç.....	240
8. SONUÇLAR VE ÖNERİLER	241
8.1 Elde Edilenler	241
8.2 Benzer İşlerle Karşılaştırma	243
8.3 Gelecekte Yapılabilecekler.....	245
KAYNAKLAR.....	249
EKLER.....	267
ÖZGEÇMİŞ.....	277



KISALTMALAR

ACL	: Access Control List
AST	: Abstract Syntax Tree
CIM	: Common Information Model
CIMOM	: CIM Object Manager
COPS	: Common Open Policy Service
DAO	: Decentralised Autonomous Organization
DARPA	: Defense Advanced Research Projects Agency
DEN	: Directory Enabled Networking
DID	: Decentralized Identifier
DMTF	: Distributed Management Task Force
DSE	: Domain Scope Expression
ES	: Event Service
GUI	: Graphical User Interface.
HPC	: High Performance Computing
HTC	: High Throughput Computing
IETF	: Internet Engineering Task Force
IP	: Internet Protocol
IPSEC	: IP Security
ISO	: International Organization for Standardization
ISP	: Internet Service Provider
JDMK	: Java Dynamic Management Kit
JESS	: Java Expert System Shell
JMS	: Java Messaging Service / Job Management System
JMX	: Java Management Extensions
JNDI	: Java Naming and Directory Interface
LDAP	: Lightweight Directory Access Protocol
MIB	: Management Information Base
MO	: Managed Object
MOF	: Managed Object File/Format
MPLS	: Multiprotocol Label Switching
OCL	: Object Constraint Language
OMG	: Object Management Group
ORB	: Object Request Broker

PBM	: Policy Based Management
PC	: Policy Controller
PCIM	: Policy Core Information Model
PDCA	: Parallel and Distributed Computing Architecture
PDL	: Policy Definition/Description Language
PDP	: Policy Decision Point
PDS	: Policy Deployment Service
PEP	: Policy Enforcement Point
PIB	: Policy Information Base
PoC	: Proof Of Concept
PPL	: Police Programming Language
QoS	: Quality of Services
RAP	: Resource Allocation Protocol
RBAC	: Role Based Access Control
RFC	: Request For Comment
RMS	: Resource Management System
RSVP	: Resource ReSerVation Protocol
SIM	: System Information Model
SLA	: Service Level Agreement
SNMP	: Simple Network Management Protocol
SPL	: Security Policy Language
SPP	: Security Policy Protocol
SRSL	: Security Requirement Specification Language
TCP	: Transmission Control Protocol
UDP	: User Datagram Protocol
VC	: Verifiable Credential
VoIP	: Voice Over IP
VPN	: Virtual Private Network
W3C	: World Wide Web Consortium
WBEM	: Web Based Management
YBH	: Yüksek Başarımli Hesaplama

ÇİZELGE LİSTESİ

Sayfa

Çizelge 2.1 : Blokzincir platform aktörleri ve sistemden beklentileri.....	29
Çizelge 3.1 : Police politika söz dizimi.	44
Çizelge 5.1 : HeteroSim metrik listesi.	114
Çizelge 5.2 : Örnek senaryo aşamalarında parametrelerin değişimi.	131
Çizelge 5.3 : Kaynakların özellikleri.	131
Çizelge 5.4 : İş yüklerinin özellikleri..	132
Çizelge 5.5 : Kullanıcı açısından metrikler (N adet iş yükü için).	133
Çizelge 5.6 : Sistem açısından metrikler.	134
Çizelge 6.1 : Kullanıcı açısından metrikler (N adet iş yükü için).	176
Çizelge 6.2 : MSVD yazılımları, verileri ve kullandıkları kaynaklar.....	184
Çizelge 7.1 : Yönetişim zorluklarına göre modellerin karşılaştırılması.	206
Çizelge 7.2 : Gerçek değişiklik önerilerinin modelimizle gerçekleştirilebilirliği.....	210
Çizelge 7.3 : Oylamada kullanılan DDO ve VC değerleri.	222
Çizelge 7.4 : DaaS çerçeveleri karşılaştırması.	236
Çizelge 7.5 : DAO Yönetişimi için önerilen yaklaşımın katkıları.....	238
Çizelge B.1 : DIDRegistryContract ve Voting Contract sözde kod gösterilimi.....	271
Çizelge B.2 : Proposal Contact oylama fonksiyonu algortiması.	272
Çizelge C.1 : Proposal Contact oylama fonksiyonu algortiması.	273
Çizelge D.1 : İkinci PoC 2 için hazırlanmış örnek politikalar (Makale içinde [10] yayınlanmış şekli ile verilmiştir).	275



ŞEKİL LİSTESİ

Sayfa

Şekil 1.1 : Çalışmada izlenen metodoloji.	2
Şekil 2.1 : MIMD mimarilerinin alt kırılımı.....	5
Şekil 2.2 : RMS mimarilerinin sınıflandırması.....	11
Şekil 2.3 : Küme yönetim sistemlerinin gelişim kronolojisi [16].....	13
Şekil 2.4 : RMS'ler için önerilen soyut bir model [18]	18
Şekil 2.5 : Globus katmanları örnek senaryosu.	23
Şekil 2.6 : Cloud katmanları özeti [28].....	24
Şekil 2.7 : BOINC mimarisi [29].....	25
Şekil 2.8 : Blokzincir sistemlerini oluşturan katmanlar.....	27
Şekil 2.9 : Blokları arası Özet Zinciri yapısı (Bitcoin).	28
Şekil 2.10 : Blokzincir sisteminde veri güncelleme akışı.....	31
Şekil 3.1 : Genel PBM mimarisi [46].	34
Şekil 3.2 : Politika tanımlama tipleri [53].	36
Şekil 3.3 : Ponder politika tanımlama dili örnekleri.....	38
Şekil 3.4 : Ponder2 çoklu PEP tasarımı.	40
Şekil 3.5 : Police sistem mimarisi.....	42
Şekil 3.6 : Nesneye dayalı Police politika modellemesi.....	43
Şekil 3.7 : Örnek yetki politikası.	45
Şekil 3.8 : Örnek emir politikaları.	45
Şekil 3.9 : Örnek SIM senaryosu ve semantik ilişkiler.	47
Şekil 3.10 : Bir RMS iş kuyruğu için model sınıfı ve semantik ilişki örneği.....	47
Şekil 3.11 : Police'in geniş bilgi modeli.....	48
Şekil 3.12 : Police bileşik olay ifadesi örnekleri ve PG olarak gerçekleşmesi.	49
Şekil 3.13 : Politika uygulamada, PC, RC ve ES arasındaki etkileşim.	50
Şekil 3.14 : Örnek PG çıktısı.	50
Şekil 3.15 : PG ile olay izleme senaryosu.	51
Şekil 3.16 : Yönetim Arayüzü üzerindeki hizmet erişim noktaları (SAPs).....	52
Şekil 3.17 : Politika Yönetim konsolu ekran görüntüleri.	53
Şekil 3.18 : POLICE PEP izleme konsolu.....	54
Şekil 3.19 : Meta-politika tabanlı çakışma kotarmanın yönetimdeki yeri.....	55
Şekil 3.20 : Police çakışma kotarma modeli.....	56
Şekil 3.21 : Ponder meta-politikası (görev ayrımı semantiği için).....	57
Şekil 3.22 : Duty separation tipinden semantik çakışma için E-Code.....	57
Şekil 3.23 : Single manager tipinden semantik çakışma için E-Code.	58
Şekil 3.24 : Sınırlı kaynak semantik çakışması için E-Code tanımlanması.	58
Şekil 4.1 : Kaynak inşaa politikası örneği.	65
Şekil 4.2 : Önerilen politika tanımlama uzayları [113].	66
Şekil 4.3 : Koalisyon ve Lokal politika formatları.	67
Şekil 4.4 : Ağ politikaları ile Grid politikalarının entegrasyonu [107].....	68
Şekil 4.5 : Grid politikası örneği [107].....	68
Şekil 4.6 : Veri Merkezleri için PBM modeli.....	69

Şekil 4.7 : Veri Merkezleri arası kaynak paylaşım akışı ve politika çeşitleri.....	69
Şekil 4.8 : İş yükü bağımlılıkları için graf ve kaynak eşleme örneği [116].....	70
Şekil 4.9 : Önerilen Kaynak Eşleyici (RM) modeli.....	70
Şekil 4.10 : SPHINX mimarisi [117].....	71
Şekil 4.11 : VO kullanıcıları ve kaynak sahiplerinin ilişkisi.	73
Şekil 4.12 : UP formatı.	74
Şekil 4.13 : VO tabanlı HPC modeli [120].....	74
Şekil 4.14 : GangSim'in bileşenleri [120].	74
Şekil 4.15 : Önerilen model [125].	77
Şekil 4.16 : Globus Şemasının Akenti Sunucu ile entegrasyonu [128].	79
Şekil 4.17 : Kullanılan iki seviyeli iş sıralama şeması [130].....	80
Şekil 4.18 : İş sıralama düğümü yapısı [130].	81
Şekil 4.19 : Kullanılan politika türleri [131].....	83
Şekil 4.20 : RMS'ler için önerilen soyut model ve gridlerde kullanımı [18].	89
Şekil 4.21 : Yönetimsel (Administrative) uzaylar arası bağlaşımlar.	90
Şekil 4.22 : [18] modeliyle gerçekleştirilebilen/gerçekleşemeyen RMS senaryoları.....	91
Şekil 4.23 : Yerel Kaynak/Kullanıcıları dikkate alan senaryo.....	91
Şekil 4.24 : Soyut RMS modeli ([18] modelinin genişletilmiş sürümü).	92
Şekil 4.25 : HeteroSim RMS düğümü ile oluşturulan YBH senaryosu örneği.....	93
Şekil 4.26 : Evrensel YBH modellemesi için kullanılan bilgi modeli.....	93
Şekil 4.27 : Evrensel YBH modelinin PBM yönetimi (örnek senaryo).....	94
Şekil 5.1 : Simülasyon motoru mekaniğine göre sınıflandırma [162].....	103
Şekil 5.2 : Simülasyon elemanları.	105
Şekil 5.3 : SimJava varlıkları arasında zamana bağlı olay dağıtımı [168].	105
Şekil 5.4 : HeteroSim'in simülatörler ile karşılaştırılması.	108
Şekil 5.5 : CloudSim mimarisi [28].....	110
Şekil 5.6 : SimGrid mimarisi [15].	111
Şekil 5.7 : HeteroSim kaynaklara atama grafiği.	114
Şekil 5.8 : HeteroSim AET metrik grafiği.....	115
Şekil 5.9 : HeteroSim AJL metrik grafiği.....	115
Şekil 5.10 : HeteroSim AWT metrik grafiği.....	116
Şekil 5.11 : HeteroSim'de kullanılan kavramlar ve ilişkileri.	117
Şekil 5.12 : SE ve RE'ler arasında AL üzerinden etkileşim.	117
Şekil 5.13 : Teknoloji-X kullanarak haberleşen SE'ler.	118
Şekil 5.14 : HeteroSim sistemi blok diyagramı.	121
Şekil 5.15 : HeteroSim gerçeklemesine ait bilgi modeli.	121
Şekil 5.16 : HeteroSim'in yüksek çözünürlüklü kaynak modellemesi.....	122
Şekil 5.17 : YBH nesnelerinin Managed Object olarak tanımlanması.	123
Şekil 5.18 : Simulasyonun başlaması akış diyagramı.....	124
Şekil 5.19 : Simulasyon yürütme ve sonlanma aşaması (Sequence diyagramı).....	124
Şekil 5.20 : Politika tabanlı görev yürütme örnek senaryosu.	125
Şekil 5.21 : Police'in HeteroSim YBH simülasyonu ile entegrasyonu.	126
Şekil 5.22 : Police PEP'in, HeteroSim'de bir RMS ile etkileşim bağlaşımı.	127
Şekil 5.23 : Örnek HPC yönetim politikası.	128
Şekil 5.24 : Doğruluk analizinde kullanılan YBH mimarisi.....	129
Şekil 5.25 : Örnek iş yükü yaşam çevrimi.	130
Şekil 5.26 : Deney1 mimarisi.	136
Şekil 5.27 : Deney1 sonucu iş-kaynak atama grafiği.....	137
Şekil 5.28 : Deney2 mimarisi.	138
Şekil 5.29 : Deney2 sonucu iş-kaynak atama grafiği.....	139

Şekil 5.30 : Deney3 mimarisi.	139
Şekil 5.31 : Deney3 sonucu iş-kaynak atama grafiği.	140
Şekil 5.32 : Deney4 sonucu iş-kaynak atama grafiği.	141
Şekil 5.33 : Deney5 mimarisi.	142
Şekil 5.34 : Deney5 Resource4-User1 erişim politikası.	143
Şekil 5.35 : Deney5 Resource4-User2 erişim politikası.	143
Şekil 5.36 : Deney5 sonucu iş-kaynak atama grafiği.	144
Şekil 5.37 : Deney5 sırasında Toplam İş Boyu (TJL) metriği grafiği.	144
Şekil 5.38 : Deney5 sonucu kısıtlayıcı politikalar ile iş-kaynak atama grafiği.	146
Şekil 5.39 : Deney5 sırasında kısıtlayıcı politikalar ile TJL metriği grafiği.	146
Şekil 5.40 : Simülasyon süresinin kullanıcı sayısına göre değişimi.	148
Şekil 5.41 : Sanal bellek kullanımının kullanıcı sayısına göre değişimi.	149
Şekil 5.42 : Maksimum Thread sayısının kullanıcı sayısına göre değişimi.	150
Şekil 5.43 : Simülasyon süresinin kullanıcı sayısına göre değişimi.	151
Şekil 5.44 : SDS Blue Gene iş izi, HeteroSim yürütmesi (TJL metriği).	153
Şekil 5.45 : SDS Blue Gene iş izi, HeteroSim yürütmesi (YBH metrikleri).	153
Şekil 5.46 : SDS Blue Gene iş izi, HeteroSim yürütmesi (AET metrikleri).	154
Şekil 5.47 : OSC Cluster, HeteroSim yürütmesi sonuçları.	155
Şekil 5.48 : OSC Cluster, HeteroSim yürütmesi sonuçları.	156
Şekil 6.1 : TARBİL Sistem mimarisi.	158
Şekil 6.2 : OpenStack servis çeşitleri.	161
Şekil 6.3 : Juno sürümü OpenStack servisleri etkileşimi.	162
Şekil 6.4 : OpenStack servisleri ve iletişim mekanizmaları.	163
Şekil 6.5 : Örnek OpenStack konuşlandırma görünümü (yeniden çizim).	164
Şekil 6.6 : IBM Cloud Manager mimarisi [203].	165
Şekil 6.7 : IBM Cloud Manager bileşen modeli [203].	166
Şekil 6.8 : Congress politika uygulama bileşen diyagramı.	171
Şekil 6.9 : Congress (Classification) politikası ve ihlal durumu örneği.	172
Şekil 6.10 : Tetris politika otomasyon bileşen örnekleri.	175
Şekil 6.11 : Tetris gerçekleştirme görünümü.	177
Şekil 6.12 : POLICE ile OpenStack PBM senaryosu.	179
Şekil 6.13 : Mevcut OpenStack token tabanlı erişim denetimi senaryosu.	179
Şekil 6.14 : TARBİL MSVD bileşenleri ve arayüzleri.	187
Şekil 6.15 : TARBİL MSVD süreç mimarisi ve öndelik ilişkileri.	188
Şekil 6.16 : TARBİL MSVD süreç mimarisi ve öndelik ilişkileri.	189
Şekil 6.17 : MSVD için tasarlanan YBH mimarisi (sadeleştirilmiş iş listesi).	190
Şekil 6.18 : ProActive İş yükü yönetim bileşenleri.	192
Şekil 6.19 : Workflow Studio: İş yüklerini bağımlılıkları tasarımı.	192
Şekil 6.20 : Resource Manager Portal: İş yükü izleme ve yönetme.	193
Şekil 6.21 : ProActiveWS-POLICE PBM Entegrasyonu.	194
Şekil 6.22 : TARBİL MSVD'nin politika tabanlı yönetilebilir mimarisi.	195
Şekil 7.1 : Blokszincir tabanlı DID çözümlerinde roller ve bilgi akışları [230].	212
Şekil 7.2 : Önerilen modelin politika yaşam döngüsü.	214
Şekil 7.3 : Önerilen Düğüm mimarisinin basitleştirilmiş bileşen diyagramı.	215
Şekil 7.4 : Öneri yaşam çevrimi (UML Sequence Diagram).	216
Şekil 7.5 : Modelimizde kullandığımız DID Mimari yaklaşımı.	218
Şekil 7.6 : Aktörlerin kayıt ettirme (güven) bağımlılıkları.	219
Şekil 7.7 : Nitelikli bakiye hesabı.	221



YÜKSEK BAŞARIMLI VE MERKEZİYETSİZ HESAPLAMA SİSTEMLERİNDE POLİTİKA TABANLI YÖNETİM

ÖZET

Yönetim sistemlerinin gelişmiş üyelerinden olan Politika Tabanlı Yönetim (PBM) sistemleri, yönetilen sistemin gerçekleştirilmesinde değişiklik yapmadan yalnız politikaları değiştirerek yönetilen sistemin davranışını değiştirmeye olanak sağlar.

Bu tez çalışmasının amacı, yüksek başarımlı sistemlerinin genelini ve onun bir alt türü sayılan merkeziyetsiz hesaplama sistemlerinin yönetim ihtiyaçlarının karşılanmasında PBM'in kullanılabileceğini göstermektir. Bu amaçla, literatürde, bu sistemlerin PBM tabanlı yönetimi konusunda yapılan çalışmalar incelenerek teknolojilerdeki mevcut durum tesbit edilmiş ve hangi açılardan geliştirme ihtiyacı olduğu anlaşılmaya çalışılmıştır. Yapılan literatür incelemelerinden, yüksek başarımlı hesaplama sistemlerinin yönetiminde gerçek anlamda PBM kullanımının, henüz emekleme aşamasında olduğu, merkeziyetsiz sistemlerde ise henüz gündeme gelmediği görülmüştür. Bu sistemlerin PBM tabanlı yönetimi için gereksinimler belirlenmiştir.

Sonrasında, belirlenen bu ihtiyaçların karşılanabilirliğinin gösterilmesi için, Police isimli PBM modeli de bu teze hazırlık kapsamında geliştirilmiş ve tezde kullanılmıştır. Police, büyük ölçekli sistemlerin politika tabanlı olarak yönetilmesine olanak veren, kendi özgün politika tanımlama diline, dağıtım modeline ve politika çakışma kotarma yeteneğine sahip olan bir PBM platformudur. Police modelinin detayları çeşitli makalelerde açıklanmıştır.

PBM kavramının yüksek başarımlı ve merkeziyetsiz hesaplama sistemlerinin yönetimi için kullanılabilirliği, tez çalışmamızda, simülasyon ve gerçek sistem örnekleri üzerinde kavram ispatı çalışması olmak üzere iki yöntemle gösterilmeye çalışılmıştır.

Birinci yöntemde (Simülasyonlar), Police platformu üzerinde PDCA (Parallel and Distributed Computing Architecture) sistemi yönetimi için gereken politikalar tanımlanmıştır. Tanımlanan politikaların etkisi, PDCA sistemlerinin simülasyonu üzerinde gösterilmiştir. Bu amaçla, evrensel, yönetilebilir PDCA modeli tasarımı yapılmıştır ve bu evrensel modele göre her türlü senaryoyu gerçekleyebilen, ayrık zamanlı, HeteroSim adını verdiğimiz simülasyon aracı geliştirilmiştir. Simülasyonlarımızda kullanacağımız politikaları belirlemek üzere yine bir literatür taraması yapılmıştır. HeteroSim'in gerçek PDCA sistemlerinin iş yüklü izlerini (workload trace) de kullanabilmesini sağlayarak, doğruluğu ve performansını değerlendirmeye çalıştık. Simülasyonlarda, belirlenmiş metrikler ölçülerek politika tabanlı yönetimin etkinliği gösterilmeye çalışılmıştır. Ayrıca, geliştirilen simülatörün ölçeklenebilirliğini göstermek için, benzer simülasyon yazılımlarında yürütülmüş ölçeklenebilirlik deneyleri, kendi simülatörümüzle de yürütülmüş ve elde edilen başarılı sonuçlar karşılaştırmalı olarak paylaşılmıştır. Simülatörümüzün doğruluğu ise tasarlanmış iş yüklerinin çeşitli senaryolar kapsamında çalıştırılıp sonuçlarının analiz edilmesi yoluyla gösterilmiştir.

Simülasyonlarımızda koşturduğumuz PDCA bileşenlerinin Police PBM çatısı ile yönetilebilmesi için, simülasyon yazılımları (PDCA senaryolarına ait bileşenler) ile gerçek dünya yazılımları (Police yazılımı bileşenleri) arasında etkileşime olanak sağlayacak bir yöntem geliştirilmesi ihtiyacı ortaya çıkmıştır. Simülasyon dünyası, gerçek dünya yazılımları ile simülasyon uzayında çalışan yazılımların, bir simülasyon oturumu kapsamında etkileşmesi için formal bir çözüm sunmamaktadır. HeteroSim'e, hem gerçek dünya yazılımlarını hem de simülasyon varlıklarını içeren oturumlar düzenlemeye olanak sağlayacak yetenek eklenmiştir. HeteroSim'in bu özgün yeteneği sayesinde, simülasyon varlığı olarak kodladığımız PDCA nesnelerini, bir gerçek dünya uygulaması olan Police yardımıyla yönetme olanağı elde edilmiştir. Böylece, Police modelinin (dolayısıyla PBM kavramının), PDCA yönetimindeki etkinliğini görebileceğimiz simülasyonlar düzenleyebilmek mümkün olmuştur.

İkinci yöntemde (Kavram ispatı çalışması) ise mevcut iki PDCA platformu üzerinde, Police PBM modeli entegre edilerek yönetim için kullanılabilirliği gösterilmeye çalışılmıştır. Seçilen bu PDCA sistemlerinden birisi İTÜ TARBİL altyapısı, diğeri yakın zamanda popüler olan Blokzincir (Blockchain) platformlarıdır.

İlk gerçek dünya PoC çalışması olarak TARBİL sistemi için, önce Bulut altyapısı ile PBM entegrasyon konuları araştırılmış ve PBM'in nasıl kullanılabileceği irdelenmiştir. Daha sonra aynı inceleme, bir JMS (Job Management System) olan ProActive isimli platform ile tekrarlanmıştır.

İkinci gerçek dünya PoC çalışmasında ise merkeziyetsiz yüksek hesaplama sistem örneği olan blokzincir platformlarının mevcut yönetim sorunları literatürden analiz edilmiştir. Daha sonra, bu sorunları politika tabanlı yönetim ile çözülebilirliği, mevcut blok zinciri platformlarının gerçekleştirmiş yönetim verilerine dayanarak analiz edilmiştir. Ardından, PBM üzerine kurulu yeni bir zincir üstü blokzinciri yönetim modeli önerilmiştir. Önerilen yönetim modeli, Ethereum blokzincir platformu üzerinde hayata geçirilmiş ve deneyimlerimiz paylaşılmıştır. PBM, anlaşılması kolay üst düzey ifadelerle yönetime olanak sağladığından, farklı bilgi düzeylerinden paydaşlara sahip blokzincir ekosistemlerine uygundur. Bu model, yalnızca mevcut zincir-dışı yönetim modellerinde daha etkili olmakla kalmaz, aynı zamanda diğer zincir üstü yönetim modellerinden daha basit ve daha kullanıcı dostudur. Ayrıca, kullandığımız merkezi olmayan kimlik kavramı, blok zinciri yönetişiminin karar verme süreci ile ilgili birçok sorunu çözme potansiyeline sahiptir. Yöntemimiz, iyi yönlendirilmiş bir yönetim sağlarken merkeziyetsizliği, güvenliği ve adaleti güçlendirir. Ayrıca, geliştirilen yönetim modelinin, DAO'ların yönetim sorunlarına getireceği çözümler de bir analiz çalışması olarak irdelenmiş ve yayınlanmıştır.

POLICY BASED MANAGEMENT IN HIGH PERFORMANCE AND DECENTRALIZED COMPUTING SYSTEMS

SUMMARY

Being one of the advanced members of management systems, Policy Based Management (PBM) systems allow to dynamically change the strategy for managing the system and hence modify the behavior of a system, without changing its underlying implementation.

The goal of this thesis is to investigate how the PBM can be used to satisfy the management requirements of general Parallel and Distributed Computing Architectures (PDCA) and Decentralized Computing Systems which can be seen as a sub category of PDCAs. Therefore, we performed a literature survey on policy based management of these systems in order to discover the state of the art and the technological issues that need to be addressed. The survey results show us that while the need for employment for decentralized systems has not yet been realized, the use of PBM in the PDCA field is still in its infancy. Next, we identified the requirements that must be met for policy based management of these systems.

Then, in order to show the feasibility of satisfying the identified requirements, we used a policy based framework, called POLICE, which was developed by the author as a preparation step to this thesis. POLICE is a PBM platform for general purpose management of enterprise-scale systems. It has an object-oriented policy specification language, a policy deployment and enforcement infrastructure and a unique policy conflict handling capability. The several aspects of this framework have already been presented in a number of articles.

In this thesis, we tried to demonstrate the efficiency and usability of policy based management approach on the management of PDCA systems and the governance of decentralized systems in two manners: through simulations and proof of concept studies on real-world systems.

In the first method (Simulations), we have presented the effect of the defined policies on simulations of PDCAs. For this purpose, we first developed a universal model of PDCA systems based on the models in the literature. Then, based on our universal PDCA systems model, we developed a discrete event simulation framework (so-called HeteroSim) that can be used to simulate any PDCA scenario. In order to determine the policy types commonly used for the management of PDCA systems, we have again performed a literature survey. We tried to make the widest version of this set of policies supported in our simulations. In order to get idea about its accuracy and performance, we added HeteroSim, the capability to run workload traces of real-world PDCAs. The designed simulation metrics make it possible to obtain quantitative results about the efficiency of the PBM. On the other hand, to demonstrate the scalability of our simulator, we ran the same experiments on HeteroSim as the research conducted on various simulators. These experiment results are also shared comparatively in the

thesis. The accuracy of HeteroSim is also analysed by running real-world workload traces in several scenarios and evaluating their outputs.

In order to manage the components of simulated PDCA with the Police PBM framework, we had to provide an integration between simulation entities (PDCA components) and real-world applications (Police enforcement elements). Simulation literature does not propose any solution for this purpose. That's why, we evolved HeteroSim to enable combining simulation and real-world software in one simulation session. We can rely on this model to build simulations that combine both simulated elements of well-known PDCA systems and already implemented elements of our PBM framework. Thus, we were able to arrange simulations in order to show efficiency of Police (and consequently efficiency of PBM concept) on the management of PDCA systems.

In the second method (organized as two Proof of Concept studies), we chose real PDCA platforms to integrate and examine the efficiency of our PBM platform. One of these selected PDCA systems is the TARBIŁ platform operated by ITÜ, and the other is the Blockchain platform, which is an example of the recently emerging distributed ledger technology.

As the first PoC (Proof of Concept) study, we focused on the TARBIŁ system. For this purpose, firstly, the transformation of the existing software structure of the TARBIŁ system, which will enable parallel processing according to the cluster model, was designed. Next, we examined the usability of the PBM concept for cloud-based conversion and management of the TARBIŁ system. We then conducted the same examination with the JMS (Job Management System) platform called ProActive.

For the second PoC, we analyzed the challenges in the Literature related to governance of Blockchain platforms (which are examples of decentralized high performance computing systems). Next, PBM adoption was analyzed based on actual governance data of existing blockchain platforms to address the issues we discovered. In this analysis, the real governance proposals of the three Blockchain platforms were examined one by one, and the feasibility of similar governance with policies was examined. After this analysis, we propose, for the first time in the literature, a new on-chain blockchain governance model that leverages both PBM and decentralized identity technologies to address the challenges posed by existing blockchain systems. In this on-chain governance model we have developed, change proposals are defined as policy. These proposals, which can be easily understood by stakeholders at every level of knowledge, are immediately put into use in the system, if accepted after a voting stage in the governance process. In the voting process of the proposals, the stakeholders can vote directly or delegate their voting rights to the delegates, if they wish, according to the liquid democracy model. As digital identities are used for onboarding stakeholders into the system and voting processes, the security of the governance process is taken to a much higher level compared to existing models. Actors must prove that they have different types of cryptographically verifiable credentials according to user types with their digital identities in the Self-Sovereign-Identity model. Another uniqueness of this model is that, thanks to the security level achieved through digital identities, actors in governance can vote with varying weights depending on their roles and the amount of digital assets in the system. Moreover, thanks to the eligible balance concept we have developed, actors do not have to lock their digital assets during the governance decision-making process.

We have realized the design of the proposed model and the implementation of the smart contracts by choosing the Ethereum blockchain platform as an example and shared our experiences in this work. The security risks of the proposed model were handled within the scope of thread analysis, and a design change was made for the detected vulnerabilities.

Since it allows controlling the behavior of the system via higher level expressions that are easy to understand, the PBM will obviously fit in well with blockchain ecosystems having stakeholders from different knowledge levels. This model is not only more effective regarding existing off-chain governance models but also is simpler and more user friendly than other on-chain governance models. Besides, the decentralized identity concept we have involved has the potential to solve many problems related to decision-making process of blockchain governance and fortifying decentralization, security and fairness while ensuring well-directed governance.

Decentralized Autonomous Organizations (DAOs), virtual companies implemented as smart contracts and living on blockchains, also need a governance model similar to the blockchain platform itself. Because DAOs have governance processes that work in decentralized ways, like the Blockchain platforms they are on.

We first examined the governance problems of DAOs in the literature in order to show that the proposed governance model can also be applied at the smart contract level. Next, we reviewed the development platforms most commonly used to implement existing DAOs. In this way, we were able to analyze how the problems in the literature occur in DAOs implemented using these platforms. After all these steps, how the proposed governance model can help to solve to the governance problems of DAOs is also handled as an analysis activity and published as an article.

As a result of all these analysis studies, we have seen that the policy-based management approach has a wide usage area in PDCA and decentralized computing platforms. Furthermore, our decentralized governance model is not only more effective than existing off-chain governance models, it is also simpler and more flexible than on-chain models.



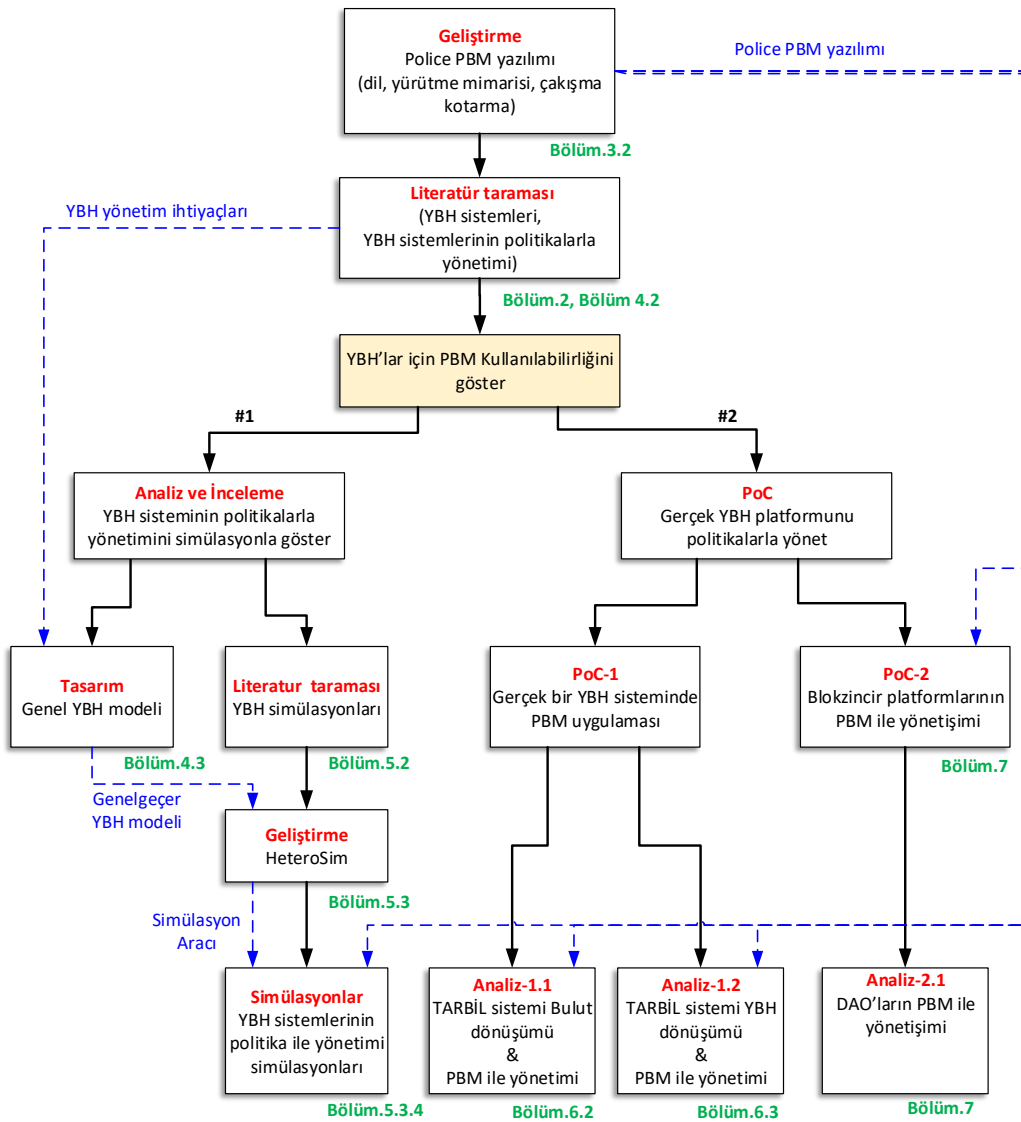
1. GİRİŞ

Literatürde PDCA (Parallel and Distributed Computing Architecture) olarak da adlandırılan, Yüksek Başarımli Hesaplama (YBH) sistemleri, genelde farklı üreticilerin ürünlerinden oluşur. Bu sistemleri yönetmek için, kullanımı kolay, ölçeklenebilir ve değişikliklere hızlı adapte olabilen yönetim sistemleri gerekir. Zira örneğin, Bulut, Grid vb. PDCA sistemlerinde, çoğu sanal olan kaynaklar, otomatik olarak ve çok hızlı oluşmakta ve yok olmaktadır. OSI'nin ağ yönetim hedeflerini (Arıza, Konfigürasyon, Envanter, Performans, Güvenlik) [1], [2] sağlamaya odaklanan geleneksel yönetim sistemleri, operatörlerin sistemdeki cihazları tek tek manuel olarak konfigüre etmesine dayalı olduklarından, YBH sistemleri büyüdükçe, yönetim ihtiyaçlarını karşılamakta yetersiz kalmaktadırlar. Mikro yönetim adımları ile büyük ve karmaşık sistemleri makro hedeflere göre hatasız yönetmek zor bir görevdir. YBH sistemlerinin yeni bir türü olan Merkeziyetsiz (Decentralized) Hesaplama sistemlerinde ise yönetim daha da zorlaşır. Sistem IT altyapısı bileşenlerinin kendi sahipleri tarafından yönetilmesinin yanı sıra, çıkar amaçlı bir araya gelmiş topluluklar tarafından, çıkarların sürekliliğini sağlayacak yönetsel müdahaleler de gerekir.

Sistemlerin yönetiminde yaşanan ve gittikçe yoğunlaşan bu zorluklar, Politika Tabanlı Yönetim (Policy Based Management) konusuna olan ilgiyi artırmıştır. Çünkü PBM, bir sistemi, teker teker sistem cihazlarının konfigüre edilmesi ile değil, hedeflenen davranışı içeren üst seviye ifadelerle (politika) yönetir. PBM, bu politikaları yorumlayıp yönetilen sistem elemanlarını, buna uygun bir şekilde otomatik olarak konfigüre eder. Böylece, yönetilen sistem bileşenlerinin kodunu değiştirmeksizin, sadece politikaları değiştirerek sistemin davranışını kısa süre içinde değiştirmek mümkün olur. Sistemin teknik detaylarını bilmesi gerekmediği için, hem operatör hatalarından kaynaklanan sorunlar engellenir hem de yönetim yükü azalır. Literatür incelemelerimiz, PBM kavramının, YBH sistemlerinin yönetim problemlerini çözebilecek potansiyelde olmasına rağmen, bu konudaki araştırmaların yeterince kapsamlı ve ürünlerin yeterince olgun olmadığını göstermiştir. Bu alandaki tüm ihtiyaçları karşılayan genel bir PBM ürünü henüz bulunmamaktadır.

1.1 Tezin Amacı

Bu tezde, yüksek başarımlı ve merkeziyetsiz hesaplama sistemlerinin yönetim gereksinimleri belirlenip sahadaki durum tesbiti yapılarak politika tabanlı yönetim yaklaşımının bu sistemlerin yönetiminde ve yönetişiminde kullanılabilirliğinin gösterilmesi amaçlanmıştır. Geliştirilen simülasyonlar, analiz çalışmaları ve gerçek dünya sistemleri üzerinde yürütülen iki Kavram İspatı (PoC) çalışmasının çıktıları kullanılarak, PBM'in sağlayabileceği faydalar irdelenmiştir. Tez çalışmalarının akışı, aşağıdaki şekilde (Şekil 1.1) gösterilmiştir. Şekildeki kutuların sağ alt köşelerinde, ilgili çalışmanın, tezin hangi bölümünde açıklandığı da belirtilmiştir.



Şekil 1.1 : Çalışmada izlenen metodoloji.

Bu amaçla, tez hazırlık çalışmasında geliştirilen Police PBM çatısı [3-8], şekilde görüldüğü gibi, tezdeki çeşitli iş paketlerinde kullanılmıştır. YBH sistemlerinin yönetiminde PBM uygulanabilirliğini göstermek için, bilinen YBH yapılarını modelleyebilecek evrensel bir PDCA modeli tasarlanmış ve bu modelin, HeteroSim adını verdiğimiz ayrık zamanlı simülasyonu gerçekleştirilmiştir [9-11]. Police PBM yazılımı gibi simülasyon dışı yazılımların YBH simülasyonlarında yönetim aracı olarak kullanılabilmesini sağlamak için, HeteroSim'e, heterojen varlık desteği de (simülasyon kodu ve gerçek dünya yazılımının birlikte çalışmasını sağlamak) eklenmiştir.

İlk PoC çalışmasında, gerçek bir YBH sisteminin PBM tabanlı olarak yönetimi sağlanmaya çalışılmıştır. Bu amaçla Police yazılımının gerçek bir YBH sisteminde (İTÜ TARBİL sistemi) pratik olarak kullanılabilirliği için çalışılmıştır. TARBİL sisteminin gerçek anlamda bir YBH sistemi haline dönüştürülmesi için iki yaklaşım geliştirilmiştir: Mevcut yazılımların YBH tipli iş güçlerine dönüştürülmesi (Analiz 1.1) ve mevcut altyapının OpenStack'e göç ettirilmesi (Analiz 1.2). Her iki öneri için de ayrı ayrı PBM entegrasyonu tasarlanmıştır. Analiz 1.2 kapsamında ayrıca, Bulut sisteminin YBH tabanlı yönetiminin tasarımı ve HeteroSim simülasyon aracının bulut sistemleri için genişletilmesi amacıyla hazırladığımız bir proje önerisi, TÜBİTAK 1003 Programında 37 Proje arasından 2. Aşamaya davet edilmiştir. Ancak ikinci aşama süreçlerine başvuru yapılamamıştır.

İkinci PoC çalışması olarak, merkeziyetsiz bir YBH türü olan blokzincirlerin yönetim (governance) ihtiyaçlarının politika tabanlı olarak karşılanması çalışması yürütülmüştür. Mevcut blokzincir platformlarının içerdiği yönetim zorluklarına yanıt vermek için iki temel teknolojiyi birleştiren yeni bir zincir-üstü blokzincir yönetim modeli önerilmiştir. Bu çalışmada, hem PBM hem de merkezi olmayan kimlik teknolojilerinin (DID: Decentralized Identifier), blokzincir yönetimi alanındaki sorunları adreslemek üzere kullanılabilirliği, mevcut platformların halihazırdaki yönetim verileri üzerinden analiz edilmiştir. Tasarlanan model, Ethereum blokzincir platformunda pratik olarak gerçekleştirilmiş ve deneyimler paylaşılmıştır.

1.2 Tezin Katkıları

Tez hazırlık çalışmasında geliştirilmiş ve tez çalışması sırasında YBH uzayına özgü yeteneklerle genişletilmiş olan Police PBM çatısı [7,8] tezin özgün katkılarından ve

hakkında çeşitli makaleler yayımlanmıştır. Tezde ayrıca, YBH sistemlerini modelleyebilecek evrensel bir model tasarlanmış ve bu model için, ayrık zamanlı HeteroSim simülasyon aracı geliştirilmiştir.

HeteroSim'e, YBH sistemi simülasyonlarında heterojen yazılım (simülasyon yazılımı ve gerçek dünya yazılımını birlikte) bulundurabilme yeteneği kazandırılması da tezin diğer bir önemli katkısıdır. Tezdeki HeteroSim'in detayları, makale olarak, SCI indeksindeki (Q2 kategorisindeki) dergide yayımlanmıştır [11].

Tezde ayrıca, mevcut blokzincirlerin karşılaştığı zorluklara yanıt vermek için iki temel teknolojiyi birleştiren yeni bir zincir-üstü (on-chain) blokzincir yönetim modeli geliştirilmiştir. Yine SCI indekslenen (Q1 kategorisindeki) Information Processing and Management dergisinde yayımlanan bu özgün çalışmada [12], merkezi olmayan kimliklere dayalı yeni bir karar alma mekanizması ve karar önerilerini politika olarak ele alma gibi bir dizi yeniliği içeren yönetim modeli geliştirilmiştir. Kavram ispatı için, Ethereum blokzincir platformunda pratik olarak gerçeklemesi de yapılmıştır. Ayrıca, geliştirilen yönetim modelinin, DAO'ların yönetim sorunlarına getireceği çözümler de irdelenmiş ve uluslararası bir konferansta yayımlanmıştır [13].

1.3 Tezin Organizasyonu

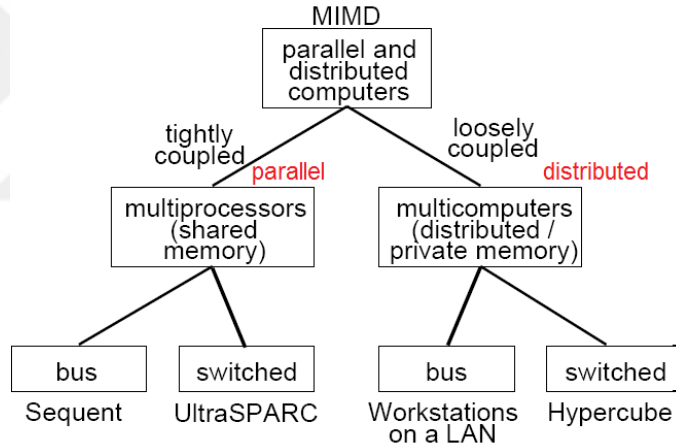
Tezin geri kalanı şöyle organize edilmiştir: İzleyen iki ana bölüm (Bölüm 2 ve Bölüm 3), sırasıyla YBH ve PBM konularında okuyucuya temel bilgileri sağlar. Bölüm 3.2 ise tezde PBM aracı olarak kullanılan Police yazılımını tanıtır. Bölüm 4, YBH sistemlerinin politika tabanlı yönetimi konusunu, yönetim gereksinimleri, literatürdeki örnek çalışmalar ve tasarlanan evrensel YBH sistemi modeli üzerinden inceler. Bölüm 5, evrensel YBH modeli için geliştirilen simülatörü tanıtır. Bölüm 6 ve Bölüm 7'de, YBH sistemlerinin politika tabanlı yönetilmesi fikrinin gerçek sistemlerde kullanılabilirliğini göstermek üzere yürütülen PoC çalışmalarının detayları verilmiştir. Bölüm 8, sonuçları literatürdeki benzer işlerle karşılaştırarak açıklar. Sonuçları ve gelecekte yürütülebilecek çalışmaları içerir

2. YÜKSEK BAŞARIMLI HESAPLAMA SİSTEMLERİ

Bu bölümde, Yüksek Başarımli Hesaplama (YBH) sistemleri ve yönetim ihtiyaçları konularında yapılan literatür araştırmasının çıktıları sunulmaktadır.

2.1 PDCA Mimari Sınıflaması

Bilgisayar mimarilerinin Flynn sınıflandırmasında [14] MIMD (Multiple Instruction Multiple Data) olarak nitelenen, Paralel ve Dağıtık Bilgisayar Mimarileri (PDCA: Parallel and Distributed Computing Architecture), yüksek başarımli mimariler olarak kabul görüp aşağıdaki (Şekil 2.1) gibi daha alt sınıflara bölünmüştür.



Şekil 2.1 : MIMD mimarilerinin alt kırılımı.

[15] makalesinde ise daha güncel bir yaklaşımla (yeni gelişen tipleri de içerecek şekilde) uygulama alanlarına göre daha detaylı sınıflara bölünmüştür:

- Super/Parallel Computing
- Cluster, Grid Computing,
- Peer-to-Peer Computing,
- Volunteer Computing,
- Cloud Computing

Paralel ve dağıtık hesaplama sistemlerinin kronolojik gelişimi aşağıdaki gibi özetlenebilir [16]:

- Başlangıçta çok işlemcili süper bilgisayarlar, MPP (Massive Paralel Processor) sistemleri üzerinde, işlemcileri program yüklerine etkin kullandırma konusunda yoğunlaşmıştır.
- Daha sonra bilgisayar kümeleri (cluster) üzerinde işlerin çalıştırılması için kaynakların paylaşılması konularında araştırmalar sürdürülmüştür (Bilgisayar kümelerinde her bir site, aynı özellikteki birkaç bilgisayardan oluşan bir MPP olarak düşünülebilir). Pahalı süper bilgisayarlar yerine daha düşük profil bilgisayarlar birleştirilerek kümeler oluşturulur. Düşük maliyetli, esnek operasyon modları sunan, CPU arızalarına karşı daha dayanıklı sistemlerdir.
- Donanım maliyetlerinin düşmesi ve PC'lerin artan performansı, bilgisayar ve bilgisayar kümelerini, bir WAN/Internet üzerinden birbirine bağlayarak kaynak paylaşımı yapmak (Grid kavramı) mümkün olmuştur. Gridlerde, önceki modellerin aksine, farklı kurumların yönetimindeki bilgisayarlar kullanılır. Kaynaklar coğrafi olarak dağıtık ve heterojendir. Dinamik davranış sergileyen kaynakların durumunu sürekli izlemek gerekir.
- Peer-to-Peer hesaplama sistemleri, bir merkezi otorite olmaksızın birlikte çalışan bilgisayarlar kümeleri olarak ortaya çıkmıştır. Bu sınıfın en güncel örneklerinden olan Blokzincirler ise özellikle Bitcoin kriptopara sistemi ile birlikte popüler olmuştur. Ortak bir hedef etrafında, birbirini tanımayan aktörlere ait bilgisayarların, bir mutabakat protokolü etrafında kümelenmesi ile çalışırlar. Çok daha yakın zamanda, blokzincirlerin birbiri ile etkileştiği (Cosmos, Polkadot vb.) daha gelişmiş blokzincir ağları da ortaya çıkmıştır.

Yukarıdaki bütün bu bilgiler ışığında PDCA platformları bu tezde, aşağıdaki şekilde kronolojik olarak sınıflanmıştır:

- Süper/Paralel bilgisayar
- Küme (Cluster),
- Bulut
- Grid
- Eşler arası (peer to peer ve decentralized) hesaplama sistemleri

2.1.1 HTC ve HPC kavramları

- YBH sistemleri, uygulama yüklerini çalıştırma şekillerine göre de sınıflanabilir. Bu amaçla aşağıdaki iki sınıf kullanılır:
- HTC (High Throughput Computing): İş yükü, N adet birbirinden bağımsız alt göreve bölünüp paralel olarak çalıştırılarak N kat hızlanma sağlanır. Parallelleştirmenin en basit hali olan bu moda, *gevşek bağlı hesaplama* veya *Grid hesaplama* ismi de verilir.
- HPC (High Performance Computing): Paralel çalışan iş yüklerinin, ortak bir amaç için işbirliği yaptığı modeldir. Haberleşen iş yükleri için MPI (Message Passing Interface) [17] defakto standart API'dir. İş Planlayıcı (Scheduler), MPI iş yüklerini düğümlere atar. Bu düğümlerde paralel işlerin başlatılması ve çalıştırılmasını ise MPI uyumlu olarak geliştirilmiş ana iş yükü yönetir.

MPI uygulamalar, bellek paylaşımı ile çalışan (shared-memory) programlara göre daha zor geliştirilse de daha fazla ölçeklenebilirlik sağlar. Cluster'lar MPI gerçekleştirme için daha uygundur. Ama eğer işbirliği yapan süreçlerin paylaşacağı veriler aşırı büyük ise bellek paylaşımı ile çalışan (shared-memory) mimari daha iyi performans sağlar.

Küme sistemleri ise genelde paylaşılan yüksek hızlı disk sistemleri içerir. Gridler ise kümelerin yüksek hızlı iletişim ağı ve yüksek hızlı paylaşılan depolama birimi olmayan sürümleri olarak görülebilir. Gridler, genelde, paralel hesaplama için tasarlanmamış donanımlar ile oluşturulurlar.

Yüksek haberleşme gerektiren paralel programlar, diğer uygulamaların da ortak kullandığı şebekeyi bloke edebileceği için Gridlere uygun olmayabilir. Gridler, çok sayıda bağımsız sürece bölünebilen ve az haberleşme gerektiren uygulamalar için uygundur (embarrassingly parallel computing).

Blokszincirler ise daha çok Gridlere benzerdir. İzinsiz Blokszincir düğümleri, merkezi koordinasyon gerekmeksizin farklı kurumlara ait bilgisayarlar üzerinde çalışarak ortak problemi çözmek için çalışırlar.

2.1.2 İş, görev, RMS ve JMS kavramları

İş yükü (Job), bir amaca yönelik işlevi yerine getiren kod parçasıdır. İş yükü, bir veya daha çok bilgisayar üzerinde çalışan, birden fazla işletim sistemi yordamı (process)

içeriyor olabilir. Bir iş yükünün içindeki alt parçalara Görev (Task) denilir. İş yükleri, şu şekilde sınıflanır:

- Basit (Batch Serial) : Tek bir CPU çekirdeğinde tek bir yordam olarak çalışır. İşin sahibinin, iş çalışırken sistemde çevrim-içi bulunmasına gerek yoktur. Paralel işlerin hazırlık/kapanışı amaçlı işler bu tiptendir. Aynı işlemciyi bir anda bir iş yükü kullandığı için hızlanma sağlamazlar.
- İnteraktif: Başlatılması ve sonuçlarının paylaşılması aşamalarında kullanıcının müdahalesini gerektiren işlerdir (örnek: program derleme). Kuyruklarda bekletilemezler.
- Paralel (Job Arrays): Aynı programın birden çok çekirdek üzerinde ve farklı girdilerle çalıştırılması için MPI vb. paralelleştirme kütüphaneleriyle üretilmiş, birbirlerine bağımlı iş kümesidir.
- Çok çekirdekli (Multicore): Diğer bütün tipleri kapsayan en karmaşık tiptir. Genelde önce basit (batch) bir iş, tahsis edilen birden çok çekirdekten birisinde çalıştırılır. Bu iş yükü, tahsis edilmiş çekirdekler üzerinde, genelde birbiri ile MPI kullanarak haberleşen yavru yordamları başlatır.

Dağıtık Hesaplama (Distributed Network Computing) uygulaması geliştirecek kişilere, kullanılan YBH sistemi mimarisinden bağımsız olarak çok işlemcili bir ortam sunmak için PVM (Parallel Virtual Machine) ve MPI (Message Passing Interface) [17] kavramları geliştirilmiştir.

YBH sistemlerinde görevlerin kaynaklar üzerinde etkin bir şekilde çalıştırılmasından ve sonuçlarının elde edilmesinden sorumlu olan yazılımlara İş Yönetim Sistemleri denilir [18]. Literatürde, Resource Management System, Job Management System, Workload Manager, Batching System, Local Resource Manager, Distributed Resource Manager ve Queuing System gibi çeşitli adlarla da isimlendirilir. Sonraki bölümlerde, bu sistemler, JMS ve RMS kısaltmalarının her ikisi ile de ifade edilmiştir.

Bilgisayar kaynaklarını verimli kullanmak için, gelen istekleri kuyruklarda biriktirip tanımlanmış politikalarla uyumlu olacak şekilde kaynaklara eşleştirip çalıştırılır. Başlatılmış işleri kaynaklar arasında taşıyabilirler (job migration). JMS'nin sağladığı kullanıcı arayüzleri üzerinden iş başlatma, durdurma, izleme, duraklatma, çıktıları alma vb. faaliyetleri yürütülebilir. Kaynak kullanımlarının istatistiğini tutarlar. Bir JMS sistemi üç ana modülden oluşur:

- Kuyruk Yöneticisi (Queue Manager): Kullanıcılardan iş isteklerini alıp depolar ve İş Sıralayıcı ile haberleşerek bu istekleri iletir.
- İş Sıralayıcı (Scheduler): Hangi işin ne zaman ve hangi kaynak üzerinde yürütüleceğine karar verir. Bu kararı verirken, iş (QueueManager'dan alınan), kaynak (ResourceManager'dan alınan) ve iş sıralama politikaları bilgilerini birlikte kullanır. Kaynak kullanımını optimize etmekten sorumludur.
- Kaynak Yöneticisi (Resource Manager): Kaynakları (konfigürasyonlarını, yük durumlarını) izlemekten, işleri kaynaklara göndermekten, kaynakları ortak kullanıma katmaktan sorumludur.

İşlerin kaynakları kullanımı iki yöntemden birisi ile gerçekleşir:

- Alan Paylaşımlı (Space sharing): Kaynak, bir iş tarafından kullanılır, diğer işler kuyrukta bekler. Kaynak, işler arasında daha az el değiştirir.
- Zaman Paylaşımlı (Time sharing): Aynı işlemcide, aynı anda birden çok süreç (preemptive olarak) çalışır. Daha yüksek öncelikli bir iş, işlemciyi, çalışan diğer işten devralabilir.

2.1.2.1 JMS fonksiyonları

Bir JMS tarafından aşağıdaki fonksiyonellikler sağlanmalıdır:

- Gönderme/Yeniden Gönderme (Re-Submission): Bir işin, tanımı, argümanları, girdi/çıkış dosyaları, çalıştırma ortamı ile ilgili tercihleri, öncelikler, uyarı tercihleri, kaynak gereksinimleri (CPU çekirdek sayısı, disk alanı, süre, işlemci süresi, bellek alanı, ağ bandgenişliği, kaynağı paylaşım yöntemi), tahsis şekli, güvenlik kısıtları vb. meta bilgilerle iş yönetimi sistemine gönderilmesi.
- Değiştirme (Alter, Rerun, Re-queue)
- Olay Bildirme (Event notification)
- Durum Kontrolü (Status Checking): İşin, *gönderildi*, *askıda*, *çalışıyor*, *tamamlandı*, *yarım kaldı* gibi durumlarını takip edilmesi.
- Durdurma/Başlatma (Holding/Suspension/Resuming)
- İşaret Gönderme (Signaling): Çalışan işe özel komutlar gönderme
- Önceliklendirme (Prioritizing)
- Muhasebe (Accounting): Kaynak kullanım muhasebesi.
- Aşama İşaretleme (Checkpointing)

- Göç ettirme (Migrating): Hata toleransı veya yük dengeleme amaçlı olarak çalıştığı kaynağı değiştirme.
- Hata Kotarma (Fault Handling)
- Özelleşmiş Başlatma (Customized Start): İş başlatma öncesi ek işlemler yapılması.
- Dosya Yönetimi (File staging-in/out): İş başlamadan önce girdi dosyalarını verme, bittikten sonra ise çıktı dosyalarını almak.

2.1.2.2 JMS tarafından sağlanması gereken özellikler

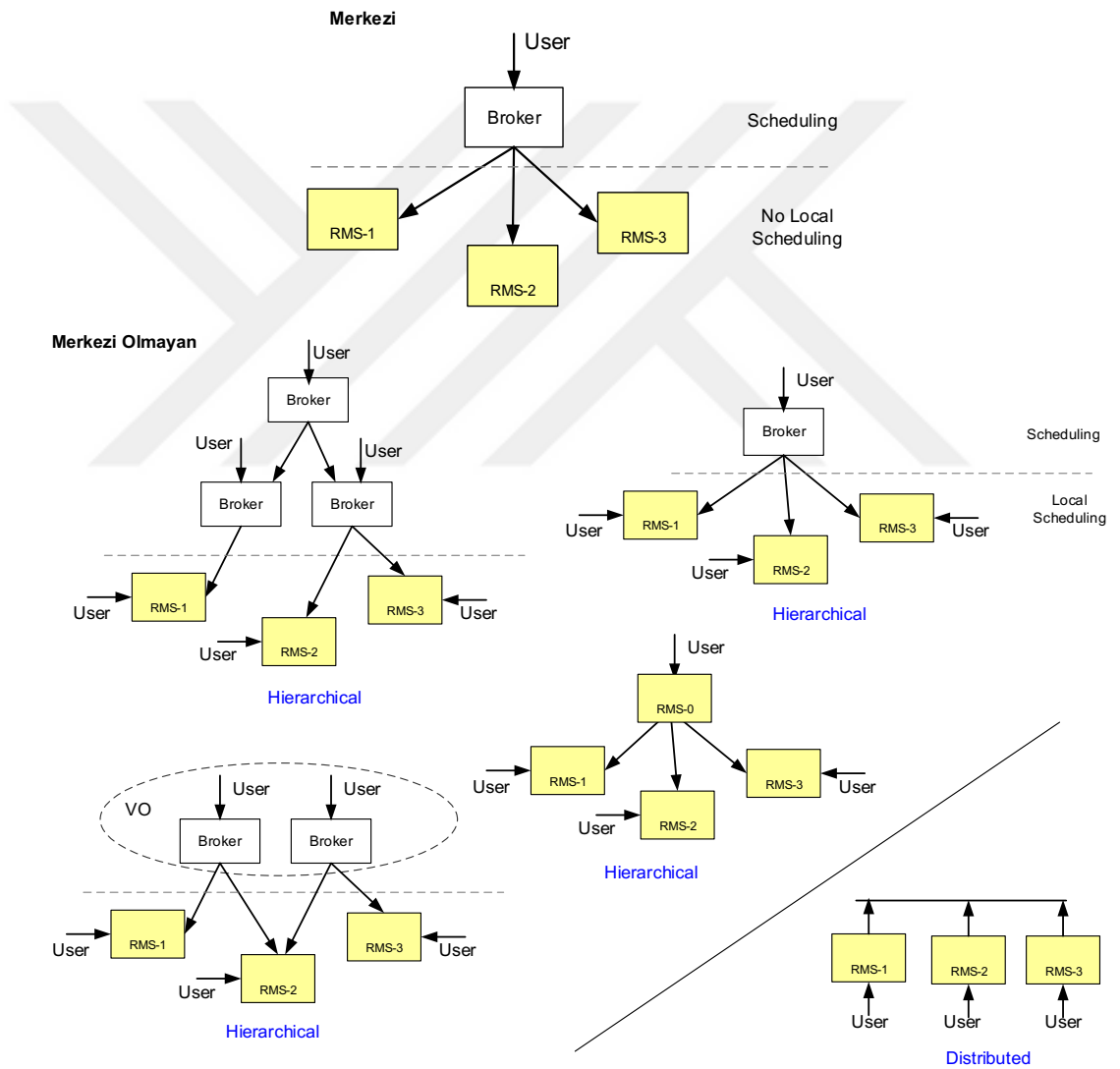
[19] ve diğer bazı makalelerde belirtilenleri harmonize ederek elde ettiğimiz, ideal bir YBH sisteminden beklenen özellikler listesi aşağıdaki gibidir:

- Kaynak kullanım isteklerinin planlanması (scheduling) yapılmalıdır,
- Kaynakları gruplar halinde tahsis edebilmeli (büyük işler için),
- Kaynakların kullanımını denetleyebilmeli, yük dengelemesi yapabilmeli,
- Sunucular veya sunucu kümeleri üzerinde, yüksek performans uygulamalarının seri veya paralel olarak çalıştırılmasını sağlamalı,
- Hem interaktif hem de toplu (batch) olarak iş çalıştırma desteği sağlamalı,
- Grafik ve multimedia türünden iş tiplerini desteklemeli,
- Farklı yönetsel etki alanlarında bulunan sistemlerin bir arada çalışabilmesi, sitelerin otonom olmasına izin vermeli,
- Çok çeşitli (heterojen) kaynakların paylaşımı (işlemci, programlar, dosyalar, veriler, bilgisayarlar ve şebekeler)
- QoS desteği (çeşitli istek ve kullanıcı sınıflarına göre değişken hizmet kaliteleri sağlamak) ve SLA desteği sağlamalı, servis kullanımını denetleyebilmeli
- Birden fazla kaynak sahibinin (otonom uzay) bulunduğu durumda, kaynakların herkesin katkısı ile orantılı olarak adil paylaşılması,
- Diğer YBH sistemleri ile birlikte çalışabilir olmalı,
- Hata-toleranslı ve sağlam olmalı,
- Adaptiflik, esneklik, modülerlik, genişleyebilirlik ve ölçeklenebilirlik sağlamalı,
- Detaylı erişim denetimi ve delegasyon desteği sağlamalı,
- Çeşitli kullanım modları (tek/çok kullanıcı, performans/maliyete duyarlı)
- Kullanıcıların kendi istekleri hakkında, kaynak sahiplerinin kaynakları hakkında, YBH sistemi yöneticilerinin ise sistem hakkında kısıtlarını dikkate almalı.

- Entegrasyonda, varolan sistemlerin yazılımlarında en az değişikliğe sebep olmalı. YBH mimarisini kullanmak isteyen (kullanıcı, kaynak sağlayıcı veya yönetici) sistemlerin varolan yazılımlarında minimum değişikliğe sebep olmalı,
- Kaynaklar üzerine yönetimden dolayı oluşan ek yük (overhead) düşük olmalı,
- İş yönetim sisteminin kendisini yönetilebilir (self-managing) olmalı.

2.1.2.3 JMS ve RMS çalışan sistemlerin sınıflandırılması

Literatürde, çok katmanlı JMS'ler oluşturmak için çeşitli PDCA yapıları kullanılmaktadır. Bu yapılar Şekil.2.2'deki çizimimizde özetlenmiştir



Şekil 2.2 : RMS mimarilerinin sınıflandırılması.

Bu yapılar temelde iki tip RMS kullanılarak oluşturulurlar:

- Aracı (Broker) RMS'ler: Kendisine ait kaynağı (hesaplama, depolama vb.) bulunmaz ve Kullanıcılarına başka RMS'lerin kaynakları ile hizmet sunar.
- Gerçek RMS'ler: Kendi Fiziksel kaynakları vardır. Örnek olarak Condor, PBS, OpenPBS, LSF, vb. küme yönetim sistemleri verilebilir.

Oluşturulan bu YBH mimarileri değişik açılardan sınıflandırılabilir. Bu mimariler, RMS'lerin aynı veya farklı yönetsel uzayda yer almaları açısından sınıflandırılabilir. Örneğin Virtual Organization (VO: ortak problem kümesi üzerinde çalışan muhtemelen coğrafi olarak dağılmış kullanıcılardan oluşur) [19] mantığıyla çalışan iki hiyerarşik RMS'ler farklı yönetsel uzaya örnektir. Bir diğer sınıflama, RMS'lerin topolojik bağlaşımları açısındandır: Merkezi, Hiyerarşik, Dağıtık RMS'ler. Örneğin, Gridlerde genelde 2 seviyeli Hiyerarşik planlama bulunur. Her iş yükü (job), önce üst seviyedeki Grid İş Yöneticisine (Global/Super/Meta Scheduler), daha sonra bir Lokal İş Yöneticisine (Local Scheduler) gönderilir. Kullanılan İş Sıralama Yöntemi de bir sınıflama kriteri olabilir. Örneğin bazı sistemlerde *ekonomi tabanlı kaynak planlaması* yapılırken çoğu sistemde sadece *QoS hedeflerine göre planlama* yapılır.

2.2 Küme (Cluster) Yönetim Sistemleri

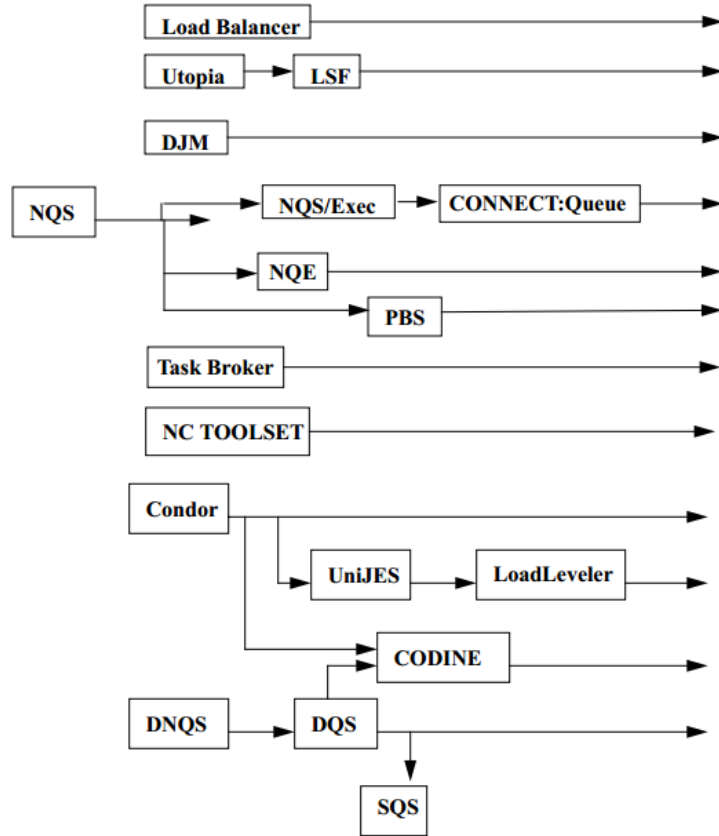
Kümeler genellikle homojen bileşenlerden oluşan daha küçük ölçekli ticari bilgisayar dizisidir. Bilgisayar kümelerinde iş yönetimi (job yönetim) için çeşitli sistemler geliştirilmiştir. Bu bölümde, bu tip JMS sistemleri kronolojik sıra ile önemli özellikleri listelenerek tanıtılacaktır.

NQS (Network Queueing System) türünün ilk örneği olarak NASA'da geliştirilmiştir (Şekil 2.3). Toplu İş Zaman Planlaması (Batch Job Scheduling) yapar. Bu akımdan, NQE, CONNECT:Queue ve PBS gibi çeşitli JMS'ler geliştirilmiştir.

LSF (Load Sharing Facility), Platform Computing firmasının ürünü olan ticari JMS sistemidir. Önemli özellikleri şunlardır:

- Kampüs gridlerini destekleyen ticari toplu iş kuyruklama sistemidir. Çeşitli lokal kümeleri aynı yönetsel etki alanı altında birleştirebilir.
- Hem HTC hem de HPC sınıfı özellikleri sağlar.
- Hem interaktif hem de toplu modda (batch) iş çalıştırabilir.
- Ardışıl ve Paralel uygulamaları destekler (Paralel iş yüklerini PVM kullanarak çalıştırır)

- Pekçok işletim sistemini destekler.
- Heterojen kümeler için gevşek bağlı küme yönetimi sağlar.
- NQS ile birlikte çalışabilir.
- LSF fonksiyonelliğini artırmak için API ve diğer arayüzler sağlar
- LSF Batch, LSF JobScheduler, LSF MultiCluster, LSF Make ve LSF Analyzer adlı modüller bulunur
- İş sıralama stratejileri: *Job Priority Based Scheduling*, *Deadline Constraints Scheduling*, *Exclusive Scheduling*, *Preemptive Scheduling* ve *Fairshare Scheduling*. Birden çok LSF iş sıralama politikası, farklı alt kümeler için paralel olarak aynı anda var olabilir.
- İş yüklerinin bir kümeden diğerine göçünü destekleyebilir.
- Kaynak tahsis stratejisi belirtmek için çeşitli mekanizmalar sağlasa da kullanıcıların QoS kısıtlarını belirtmelerine olanak sağlamaz.



Şekil 2.3 : Küme yönetim sistemlerinin gelişim kronolojisi [16].

Condor [20] adlı sistem, diğer JMS yazılımlarını (IBM LoadLeveler, CODINE vb.) tetiklemiştir. Condor'un önemli özellikleri şunlardır:

- HTC tipinden JMS uygulamasıdır. Condor, kümeler arasında kaynak paylaşımını da yönetebilir ama Gridler için daha uygundur. Globus Grid sistemi ile entegrasyon için çeşitli modüller içerir.
- (Long-running) iş yüklerinin boş bilgisayarlarda çalıştırılması için tasarlanmış dağıtık bir toplu iş yönetimi (batch) sistemidir.
- Adil paylaşım (fair-share) stratejisine göre çalışır.
- Kaynak yönetimi, arz-talep eşleştirme (matchmaking) mekanizması ile yapılır. Condor, kaynak sahiplerinin *Class-Ad (classified advertisement)* ilanlarını kullanarak iş yükleri ile uygun kaynaklar arasında eşleme yapar.
- QoS desteği içermez.
- Ardışıl ve Paralel işleri yönetebilir. Kaynak tahsisinde, kaynak sahiplerinin kısıtlarını dikkate alır. Kaynak havuzlar düz (flat) hiyerarşidedir.
- Farklı sahiplikteki çok sayıda kaynağı yönetebilir. Kaynakların bağlı olduğu yönetim uzayından bağımsız olarak, dağıtık kaynaklar havuzundan oluşan tek bir sistem görüntüsü oluşturur.
- Her kaynak üzerinde bir *Condor ResourceAgent* çalışır ve kaynaklarını periyodik olarak *Condor Collector*'e duyurur. *Condor Matchmaker*, bu bilgileri kullanır. Kaynak istekleri ve kaynak sahiplerinin kaynak paylaşımları, *Condor ClassAd* dilinde ifade edilir.
- İş yükü kuyruklama, iş sıralama politikası, öncelik şeması, Durum İşaretleme (check-pointing) ve iş göç ettirme, uzak sistem çağrılar, kaynak gözleme ve kaynak yönetimi sağlar.
- Condor'un, Globus ile çalışacak şekilde genişletilmesi ile Condor-G adlı, global Gridlerin oluşturulmasına ve iş yüklerini farklı yönetimsel etki alanlarında çalıştırabilmesine olanak sağlayan sürümü elde edilmiştir.

HTCondor, PBS ve LSF'ye benzer. Paralel iş yükleri için MPI ve PVM'i desteklese de HPC küme sistemi için değil, gridler için geliştirilmiştir. Heterojen kaynakları birleştiren bir HTC sistemi olarak geliştirilmiştir. HTCondor, hem kaynak sahipleri hem de iş göndericiler için ClassAd mekanizması içerir. HTCondor, programı ve girdi dosyalarını bütün düğümlere otomatik olarak kopyalar, iş bitince de çıktı dosyalarını otomatik olarak taşır. Kaynak kodları, hedef bilgisayara taşınıp derlendikten sonra çalıştırılır. Ortak dosya sistemi olan şebekelerde de çalışabilir. Bu tür sistemlerde büyük dosya kullanan iş yüklerinde darboğaz oluşmasını önlemek için dosyalar, işler

çalışmaya başlamadan önce düğümlerin disklerine kopyalanır (prestaging). Ancak düğümlerin bu yerel diskleri, merkezi diskler kadar hızlı değildir ve bilgisayarın yerel uygulamaları ile de paylaşılır. HTCondor, bağlantılı işlerin sıralama tasarımı için DAG (Directed Acyclic Graphs) destekler.

Açık kaynak kodlu ve ücretsiz olan HTCondor, Linux, Unix, Mac OS X, FreeBSD ve bütün Windows işletim sistemlerini destekler. NASA Advanced Supercomputing facility (NAS), 350 düğümlü bir HTCondor sistemidir.

HTCondor-G, gridler ve bulut sistemleri ile çalışma için geliştirilmiştir. Torque/PBS ve LSF gibi diğer küme sistemleri ile de çalışabilir. Globus Toolkit tarafından da desteklenen bir platformdur. HTCondor, HPC sistemlerinde istemcilere sunulan servis arayüzünde ortak bir standart oluşturmak için geliştirilmiş olan DRMAA job API'yi de destekler.

CODINE'in (Computing in DIstributed Networked Environment) gelişmiş versiyonu olan GRD (Global Resource Director)'nin önemli özellikleri ise şunlardır:

- Dinamik iş planlama/sıralama ve kaynak yönetimi
- Dinamik performans verisi toplanması
- Üst düzey politika yönetimi
- Hem interaktif hem de toplu (batch) iş yönetimi
- Dört iş sıralama politikasının (shared-based, functional, initiation-deadline, override) kombinasyonları ile iş yükü yönetimi stratejisi tanımlanabilir.

PBS, NASA için geliştirilmiş ve çok yaygın kullanılan, POSIX uyumlu, açık kaynak kodlu, JMS yönetimi sistemidir. Önemli özellikleri:

- Kuyruklama ve toplu (batch) iş yüklerinin heterojen küme kaynaklarında yürütmesini destekler.
- Toplu iş çalıştırma konusunda iyi ama gridlerden ziyade kümelere uygundur.
- Hem interaktif hem de toplu (batch) modda iş çalıştırabilir
- Toplu iş yükleri arasında bağımlılık tanımlanmasına olanak sağlar
- Konfigüre edilebilir iş sıralayıcısı bulunur
- Otomatik Dosya yönetim desteği sağlar.
- ACL (Access Control Lists) ile güvenlik sağlar
- Kompleks iş sıralama politikaları, C, Tcl script dili veya BaSL kullanarak hazırlanabilir.

- Çeşitli kaynak tahsisi stratejileri sunar (*Round-Robin, First In First Out, Load balancing, Priority based ve Dedicated times*).
- PBS'te ön tanımlı iş sıralayıcı, CPU kullanımını maksimize eden FIFO'dur.
- Altair firması tarafından devralınmış ve OpenPBS ile PBSPro adlı iki ürüne dönüşmüştür. OpenPBS, akademik amaçlı açık kaynak kodlu, kendi politika dili olan platformdur. Windows OS desteği, Grid desteği, gelişmiş iş sıralama vb. daha gelişmiş yetenekler ancak PBSPro sürümünde bulunur.

TORQUE (Tera-scale Open-source Resource and QUEue manager), Cluster Resources şirketi tarafından OpenPBS'in gelişmiş, popüler bir sürümüdür. Genelde açık kaynak kodlu Maui ve ticari MOAB genişletmeleri ile birlikte kullanılır. Dünyaca yaygın gridlerde en çok kullanılan JMS yazılımıdır. Küçük ve orta büyüklükteki küme sistemlerinde de çok yaygın kullanılır. PBS'nin üzerine çok sayıda yama ve hata düzeltilmesi yapılarak olgun bir ürün haline dönüşmüştür. TORQUE/Maui, geniş gridler için ilk tercih olarak yer bulmaktadır.

Maui iş sıralayıcı, küme ve süper bilgisayarlar için bir açık kaynak kodlu, harici iş sıralayıcıdır. Diğer JMS yazılımları (OpenPBS, PBSPro, Torque, SGE, LSF ve Loadleveler) içerisinde, politika motoru olarak iş sıralayıcı rolünde kullanılabilir. İşlere optimize bir şekilde kaynak tahsisi yapar. Önemli özellikleri şunlardır:

- Agresif iş sıralama politikaları kullanarak kaynak kullanımını optimize etmek ve iş yükü yanıt süresini minimize etmek amaçlanır.
- Çok sayıda iş sıralama politikası, dinamik öncelikler ve rezervasyona konfigüre edilebilir destek sunar,
- Site yöneticileri kaynakların iş yüklerine ne zaman ve nasıl tahsis edileceğini politikalarla yönetebilir
- İş yükü *önceliklendirme, tahsis, rezervasyon ve iş sıralama politikaları* vardır.
- QoS desteği ile kullanıcılara ayrıcalık verilebilir (ek kaynak, öncelik)
- Gelişmiş bir Rezervasyon altyapısı sayesinde siteler, kaynakların ne zaman, nasıl ve kim tarafından kullanılacağını belirleyebilirler. Her Rezervasyon üç bileşenden oluşur: kaynaklar, zaman dilimi ve erişim denetimi listesi (ACL).
- Rezervasyon ve dinamik öncelikler sağlasa da işleri gridde bağımsız kaynaklar üzerinde yürütme konusunda sınırlamaları vardır.
- İş sıralama politikası tanımlama için esnek mekanizmalar sağlar.

- Desteklediği iş sıralama yöntemleri: *fair share*, *backfilling*, *fixed resource reservation*, *iş karakteristiklerine göre önceliklendirme*.

Sun Grid Engine (SGE), Sun firmasının YBH aracıdır. Önceki adları CODINE (COmputing in DIstributed Networked Environments) ve GRD (Global Resource Director)'dir. En çok kullanılan ücretsiz JMS yazılımındandır. Kuyruk yönetimi karmaşıktır. MPI desteği çok gelişmiş değildir. DRMAA arayüz desteği bulunur.

Sun firmasını satın alan Oracle ücretsiz SGE desteğini keserek Univa firmasına satmıştır. SGE ile ilgili üç farklı ve bağımsız koldan (Univa Grid Engine, Open Grid Scheduler, Son of Grid Engine) çalışmalar devam etmektedir.

SGE, küçük ve orta büyüklükteki hesaplama merkezleri için bir tercih olabilir. Bir politika modülü, kullanıcıların, global kullanıma açtığı kaynaklar oranında kümeden faydalanmasını sağlar. Kullanıcılara kaynak kullanım biletleri atanır. Politika, ancak aynı yönetimsel alanda bulunan kampüs gridlerine uygundur. Farklı kaynak paylaşım politikaları ve QoS kısıtları olan kullanıcılar olduğunda çok faydalı değildir.

SLURM (Simple Linux Utility for Resource Management), Açık kaynak kodlu, yüksek ölçeklenebilirlik sağlayan popüler platformdur. Eklentilerle genişletilebilir altyapıya sahiptir. Maui gibi iş sıralayıcılarla entegre olabilir. Linux ve Unix platformlarında çalışır. Diğer iş sıralama yazılımlarından yüksek başarımlı, ölçeklenebilirlik modülerlik ve hata dayanıklılığı yönlerinden ayrışır.

Maui ve LSF, küme seviyesinde iş sıralayıcı içerirler. İş kuyrukları, kullanıcı, grup, düğüm, iş sıralama stratejileri yönetilen bileşenlerdir. LSF sadece hesaplama amaçlı olarak CPU paylaştırır. Depolama, yazdırma, ağ bant genişliği, vb. daha geniş yelpazedeki diğer kaynak tiplerinin yönetimini yapmaz.

En popüler üç JMS ise LSF, PBS, CODINE/GRD'dir. Bunların özellikleri [21] çalışmasında irdelenmiştir.

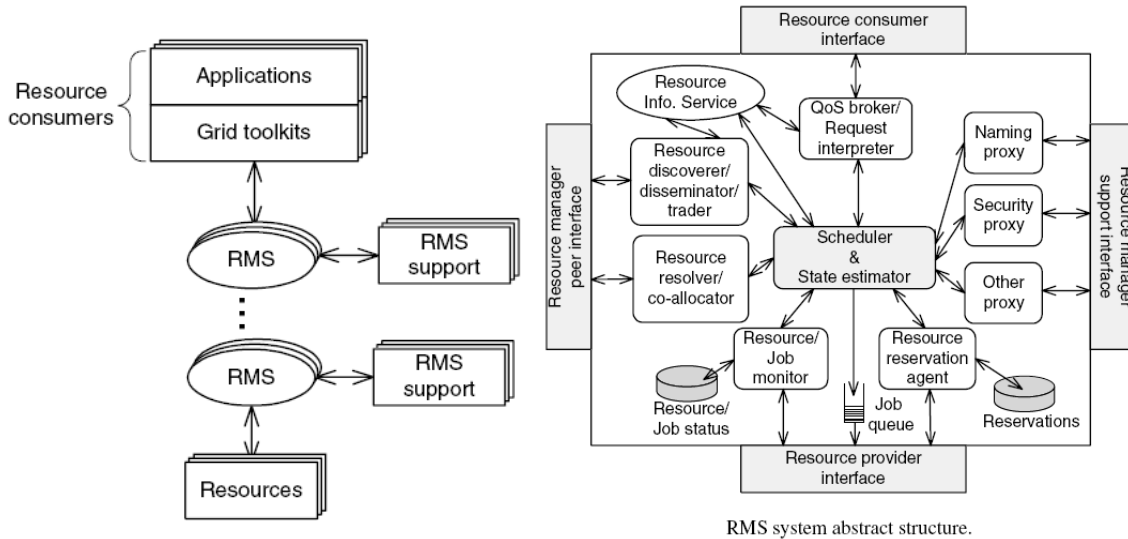
2.3 Gridler

Farklı kişi/kurumlara ait makinalardan oluşan, çok büyük ölçekli (Internet çapında) dağıtık ve paralel hesaplama sistemleridir. Milyonlarca çekirdekli dev bir bilgisayara benzeyen, Dünya geneline yayılmış kümelerden oluşan Grid'ler, bilimsel büyük problemlerin çözümü çalışmalarında ve ticari dünyada kabul görmüştür. Gridler [22] sayesinde farklı kurumlar ve farklı coğrafyalar arasında heterojen kaynakların (sunucu, dosya sistemi, veritabanı, uygulama vb.) paylaşımı mümkün olur. Virtual

Organization (VO) [19] terimi, ortak problem kümesi üzerinde Gridde çalışan, coğrafi olarak dağılmış kullanıcılar için kullanılır. VO, kullanıcıları, rollerini ve çeşitli özelliklerini tanımlar; kaynak sahipleri ile üyelerin erişimi konularını düzenler. Hesaplama, Veri ve Servis amaçlı gridler vardır. Grid bileşenleri şunlardır:

- Kaynak (Resource): Zaman planlaması yapıp kullanılması sağlanacak varlıklardır. Her kaynak tipinin bir takım statik özellik bilgileri (adı, işlemci mimarisi, hızı, sayısı, işletim sistemi vb.) ve dinamik özellikleri (durumu, anlık işlemci yük durumu, bellek kullanımı, boş disk alanı, vb.) vardır.
- İş/Görev (Job): İş istekleri içinde QoS gerekleri de belirtilebilir. İşler, isteklerinin karşılanma şekli açısından iki sınıfa bölünür:
 - *Rigid Paralel*: Baştan belirtilen miktarda kaynak üzerinde çalıştırılır,
 - *Mouldable*: Sayısı yürütme anında belli olan kaynaklar üzerinde çalışır.
- Kaynak sağlayıcı: Kaynağın sahibi
- Kaynak tüketici: Kaynak kullanan kullanıcı
- Kaynak Yönetim Sistemi: Kaynak havuzunu yöneten RMS

Grid bileşenleri için detaylı bir ontoloji, [18]'de önerilmiştir (Şekil 2.4). Gridlerde çok katmanlı RMS kullanımı olabilir (Örneğin uygulama seviyesi iş sıralama aracı olarak AppleS, Kaynak Planlayıcı olarak ise Globus).



Şekil 2.4 : RMS'ler için önerilen soyut bir model [18]

RMS sistemlerinin bir sınıflandırması, aşağıdaki kriterlere göre verilebilir [18]:

1. Grid içi makina organizasyonu
 - Merkezi ve tek denetleyici
 - Düz hiyerarşi (her maina birbirine eş)

2. Kaynak modelleme (Grid kaynaklarının uygulamalarca nasıl tanımlandığı)
 - Şema tabanlı: Bir kaynak, bir tanımlama dili ve bir dizi bütünlük kısıtları yardımıyla tanımlanır (Örneğin *Condor ClassAd* şeması).
 - Nesne modeli: Kaynaklar üzerindeki operasyonlar da kaynak modelinin bir parçası olarak tanımlanır (Örneğin *Legion* yaklaşımı).
3. İsim uzayı organizasyonu (ilişkisel, hiyerarşik, graf tabanlı)
4. Kaynak Durum bilgisi dağıtımı
 - Batch/periodyk: Kaynak bilgisi biriktirilir ve izleyenlere periyodik olarak gönderilir (push/pull). Örnek: Condor ve European DataGrid.
 - Online/OnDemand: Bilgi, oluşur oluşmaz ve sorgulandığında gönderilir.
5. Veri depolama organizasyonu,
 - Dağıtık nesneler,
 - Dizin veri depoları (Örnek: Globus MDS)
6. Kaynak keşfi (query, agent based),
 - Dağıtık, kaynak keşif ve tahsis sistemi (küçük işlere uygun).
 - Kaynakların ve kaynak isteklerinin merkezi bilgi bankasında tutulması büyük ve periyodik işlere uygundur. Merkezi ve Lokal iş sıralayıcıların olduğu hiyerarşik yapılarda kullanılır.
7. QoS desteği, Globus içinde sınırlı bir kaynak rezervasyon desteği vardır. Şebeke band genişliği, vb konularda QoS desteği veremez.
8. İş sıralayıcı organizasyonu
 - Topolojiye göre
 - Merkezi: Tek denetleyici, ölçeklenemez, çoklu otoriteyi ve çok yerden politika tanımlamayı desteklemez, hataya duyarlı.
 - Dağıtılmış: Kaynakların bazı iş yüklerine eş tahsisine (co-allocation) destek yok, yönetimi zor, (Örnek: Bond, Ninf)
 - Hiyerarşik: Çok yerden politika tanımlamayı desteklemez (Legion)
 - Delongi sınıflandırması [23]
 - Lokal
 - Global
 - Statik
 - Dinamik (Merkezi veya Dağıtık)

Gridlerde genelde iki seviyeli planlama kullanılır. Üst seviyedeki Grid/Global İş sıralayıcı (GS) üzerinden Lokal İş Sıralayıcıya (LS)'lara doğru iş akışı vardır. Her LS'nin kendi yerel iş sıralama politikası vardır. GS'nin, LS'ler üzerinde kontrolü yoktur. LS'ler kendi lokal politikalarını uygulamakta serbesttirler ama dağıtık hesaplama kaynakları üzerinde kontrol hakları yoktur. Grid kaynaklarının keşfi, kullanımlarının izlenmesi ve iş yüklerinin lokal kuyruklara yerleştirilmesi, GS'nin görevidir.

9. İş sıralama politikası

- Sabit politika: Kaynak yönetim politikası önceden tanımlıdır. Sistem veya Kullanıcı metrikleri maksimize etmeye çalışır. (AppleS).
- Genişletilebilir politika: Dış varlık tarafından iş sıralama stratejisi değiştirilebilir. Örneğin işin sırasına veya RMS davranışına müdahale edilebilir (Legion yaklaşımı).

10. Durum Kestirimi (State Estimation)

- Non-predictive: Geçmiş bilgilerini hesaba katmadan sadece o anki iş ve durum bilgisi kullanılır.
- Predictive: Durum tahmini için geçmiş ve şimdiki bilgileri hesaba katar. Ancak sistem boyutu nedeniyle güncel bilgileri toplamak her zaman mümkün olmayabilir.

11. Kaynak paylaşırma

- Zaman-paylaşımlı (time-sharing): Aynı işlemciler farklı zaman dilimlerinde farklı uygulamalara tahsis edilir.
- Alan-paylaşımlı (space sharing): İşlemci kümeleri her uygulamaya izole olarak atanır. Alt sınıfları: *statik*, *adaptif* ve *dinamik* alan paylaşımıdır.
- Hibrit: Üsttekilerin birleşimi.

12. İş Sıralama. İş gelir gelmez çalışacağı kaynaklar seçilir (on-line) veya gelen işler biriktirilerek birlikte planlanır (batch). İş yükü seçim süreci dört durumda çağrılır: yeni iş yükü geldiğinde; iş yükünün yürütmesi sonlandığında; çalışan iş yükü kesintiye uğradığında (interrupt) ve tekrar yüklendiğinde; her bir iş yükü planlama çevrimi başlarken. [24]'de, İş Sıralama yöntemleri iki ana sınıfa ayrılır:

- Kuyruk kullananlar (batch)
 - İş yükleri bir kuyruğa gönderilir (NQE, PBS, LSF vb.). Kuyruklar, kullanılabilir kaynaklara göre sınıflandırılır. Tahsisçi, isteği kuyruğun kategorisine göre çalıştırır.

- Dinamik iş sıralama olarak (tek sahibi olan kümelerde) 3 model kullanır: *FCFS (First Come First Served)*, *SJF (Shortest Job First)*, *LJF (Longest Job First)*. Kuyruktaki isteklerin ortalama cevaplanma süresine (ART) göre iş sıralama stratejisi birinden diğerine değiştirilebilir.
- Kümeler için iki aşamalı Hiyerarşik İş sıralama planlaması yapılır. İlk aşamada GS (Global Scheduler) iş yüklerini planlar ve lokal iş sıralayıcılara atar. İkinci adımda yerel iş sıralayıcı, en güncel kaynak verisini kullanarak GS'nin yaptığı planlamaya ince ayar yapar.
- Match-making (Online/On-demand): Kaynak seçimi, *İlk-Uygun (First-Fit)* veya *En-Uygun (Best-Fit)* yaklaşımı ile yapılır. Farklı kaynak sahiplik uzayları varsa uygundur. Dezavantajı, işleri birbirinden ayrı, tek tek planlanmasıdır. Bir işi en uygun kaynağa atama veya bir kaynağa uygun bir iş bulma, bu atamadan dolayı kuyrukta bekleyen bir sonraki işin kaynak bulamamasına neden olabilir.

13. İş sıralama (Scheduling): makine seçim stratejisi olarak aşağıdakiler kullanılır:

- *FirstFit*. İlk uygun makineye iş atama
- *BestFit*. Atamadan sonra sistemde boşta EN AZ sayıda düğüm kalır.
- *Backfilling* (Lifka [25]). İhtiyaç duyulan tahmini yürütme süresi bilgisini kullanır. Eğer sıradaki iş, yeterli kaynak olmadığı için başlatılamıyorsa, boştaki kaynaklara uygun ve daha sonraki işleri geciktirmeyecek başka bir iş yükü çalıştırılır. Sadece bir sonraki iş geciktirilmeyecek şekilde seçim yapılır (*Easy backfill*) veya Kendisinden önce sisteme gönderilmiş hiç bir işin tamamlanma süresini uzatmayan bir iş seçilir (*Conservative backfill*).
- Geçmişte en hızlı iş bitiren makineye, en kısa kuyruk boylu makineye, en az yüklü makineye (*dinamik*), *random(statik)*, *round-robin(statik)*.
- En az çökmüş makineye iş atama. LUNF (Longest Uptime Node First)
- İşin karakteristiklerine göre atama
- Politikalara, iş/kaynak kısıtlarına göre dinamik makina seçme

Bu tezde, hem kuyruktan iş seçmede hem de işe kaynak atamada, politikaların kullanılması seçeneği çalışılmıştır. Politikalar içinde, hem işlerin ve kaynakların bütün (statik/dinamik) özelliklerini hem de iş ve kaynak sahiplerinin kısıtlarını ifade edip iş

sıralamasında kullanmak mümkün olur. Politika tabanlı iş sıralama, statik olan diğer modellere göre daha dinamik, daha esnek ve anlaşılması kolay bir modeldir.

[18]'de Globus, AppLeS, Nimrod/G, Bond, Legion, Condo-G, European Data Grid vb. çok bilinen grid altyapıları karşılaştırmalı olarak incelenmiştir. İncelenen sistemlerin çoğu hesaplama ve servis gridleridir. Nimrod/G ve AppLeS, Globus ve Legion gibi diğer Gridlerle entegre olabilen Grid iş sıralayıcılarıdır.

Genişletilebilir iş sıralama politikasına sahip, dağıtılmış, çevrim-ichi iş sıralama sistemi, Gridler için henüz geliştirilmemiştir.

Globus [26], grid tabanlı sistem oluşturmak için en bilinen araçtır. Gerekli temel YBH servislerini (kullanıcı ve kaynakların kimlik doğrulaması, kaynak saptama/atama, kaynak yönetimi, veri yönetimi, kaynak rezervasyonu, iletişim, vb.) sağlayan GMT (Globus Metacomputing Toolkit) adlı bir araç kümesi sağlar. Kendisi kaynak tahsis kararlarını vermez. Globusun sağladığı temel servisler kullanılarak, paralel programlama, kaynak keşfi ve iş sıralayıcılar gibi daha yüksek seviyeli hesaplama servisleri oluşturulabilir. Globus, MDS (Metacomputing Directory Service) adlı dizin servisi içerir. Bu servisteki bilgiler, NIS, SNMP gibi tekniklerle derlenir. LDAP tabanlı olan MDS iki servisten oluşur:

- *Grid Resource Information Service* (GRIS) kaynak keşif servisi. Kaynak bilgi sağlayıcılar, GRIS'teki bilgileri periyodik güncellerler.
- *Grid Information Index Servers* (GIIS) pek çok GRIS'ten bilgileri çekerek grid kaynaklarının global görüntüsünü oluşturur.

Globus, kaynak rezervasyonu sağlar. Önceden tanımlanmış Globus iş sıralama politikaları, Nimrod/G, AppLeS ve Condor/G gibi uygulama seviyesi iş sıralayıcılar kullanılarak genişletilebilir.

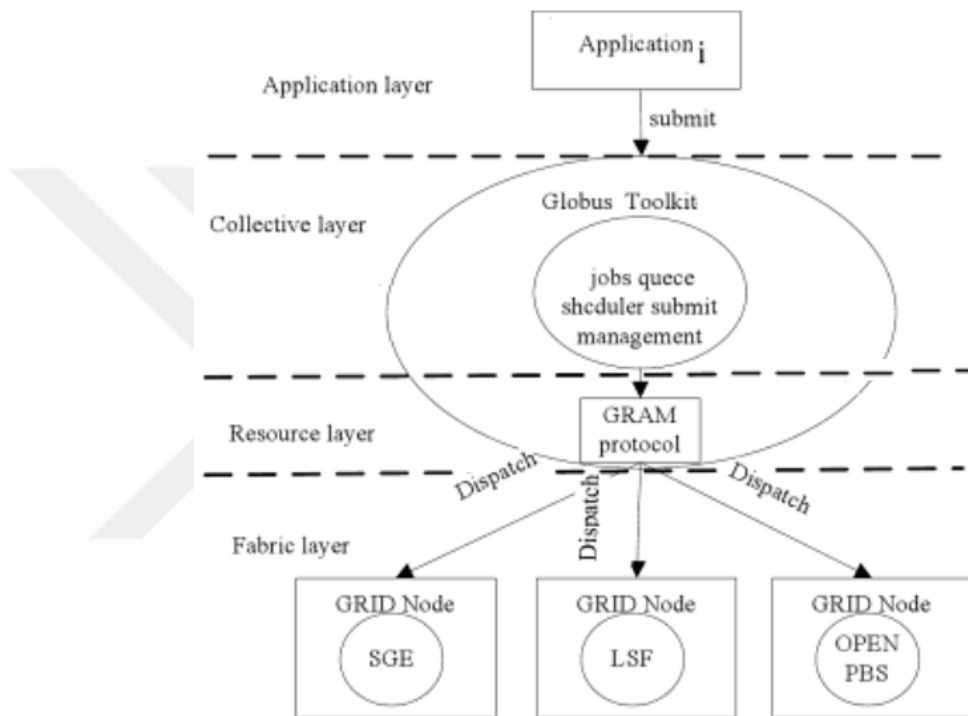
Globus modülleri, kendilerini kullanan uygulamalara, ilgili servisin kullanılabilmesi için bir dizi gerçekleştirme seçeneği sunar. Seçimler kural tabanlıdır (Örneğin haberleşme modülü için "use TCP/IP if load is low else ATM"). Uygulamalar, belli koşullar gerçekleştiğinde kendilerinin çağrılmasını (callback) isteyebilirler.

Globus Katmanlı Grid modeli (Şekil 2.5), aşağıdaki bileşenlerden oluşur:

- Fabric Layer, grid protokolleri ile erişilebilecek kaynakları sağlar.
- Connectivity Layer, grid şebekesinde gereken iletişim ve kimlik doğrulama protokollerini sağlar (PKI tabanlı protokoller).

- Resource Layer, kaynaklar üzerinde, güvenli pazarlık, başlatma, izleme, denetim, muhasebe ve ödeme operasyonlarını gerçekleştirir.
- Collective Layer, tekil kaynakları değil kaynak kümeleri ve aralarındaki etkileşimi bilir. GIIS üzerinde kaynak kümelerinin çeşitli görünümleri tutulur. GRIS'lerden kaynak durumlarını öğrenir. Verileri Gridde çoğullayabilir.
- Application Layer, VO uzayında çalışan kullanıcı uygulamalarıdır.

Globus Toolkit'in grid katman modelindeki yerleşimi, literatürde sıklıkla kullanılan aşağıdaki şekilde verilmiştir.



Şekil 2.5 : Globus katmanları örnek senaryosu.

Javelin, Paralel hesaplama için Java tabanlı altyapı sunan hiyerarşik RMS'li hesaplama grididir. Sabit ve uygulama uzayına özel iş planlama politikası kullanır.

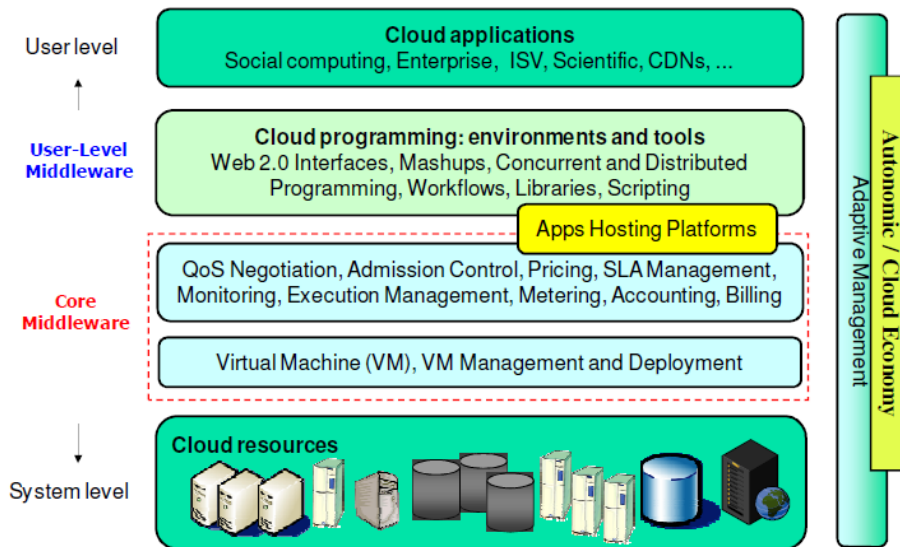
Legion, kaynakları nesneler, aralarındaki etkileşimi ise nesne metod çağrılarları şeklinde modeller. Kaynak rezervasyonu ve uygulama seviyesi (application level) iş sıralayıcı kullanımını destekleyebilir. Hiyerarşik ve dağıtık iş sıralayıcı içerebilir.

Nimrod/G, hesaplama gridleri için bir Grid aracısıdır (broker). Nimrod/G, grid kaynak yönetimini ekonomik teorilere dayanlı bir iş planlama bileşeni olarak kullanılır. Kullanıcı-tanımlı iş sıralama politikaları oluşturmaya izin verir. Kaynak keşfi ve paylaşımı için Globus ve Legion gibi sistemlerin sunduğu servislerden yararlanır. Durum kestiriminde kronolojik fiyatlandırma bilgilerini kullanır. İş planlama politikası, kullanıcının zaman ve bütçe gibi kısıtlarını içerebilir. Bazı ortak kısıtlar,

farklı kişilere ait heterojen kaynaklı ortamlarda, kümenin durumunda ve yükte dinamik değişiklikler oluşabiliyorsa (düğüm ekleme, çıkarma vb), tek tür tahsis politikaları ile performansı artırmak zordur. NP-complete olan optimal çözüm yerine tüm çözüm kümesini araştırmayan bir sub-optimal çözüm kabul edilmelidir. Yapay zeka grubu çözümler optimal çözümü arayıp bulduktan sonra kaynak atamayı yapar. Gridlerde optimal kaynak seçenekleri bulmada, LP (Linear Programlama) veya Evrimsel (Evolutionary) Algoritmalar (EAs) kullanan çalışmalar da vardır. EA'lar çözüm seçenekleri üzerinde rasgele arama yaparlar ve girdi olarak, bir dizi iş için kaynak gereksinimleri, öncelikleri, tercihleri, şimdiki kaynak durumu ve kaynak/sitenin politikaları alırlar. Bir genetik algoritma, işlerin yürütme sırası için en iyi sıralamayı bulmaya çalışır. Bu amaçla, her bir kullanıcı için otomatik olarak (çevrim içi bir evrimsel öğrenme süreci ile) elde edilen ve işler için gereken gerçek kaynak miktarını tahmin eden bir uyum fonksiyonu kullanılır. Bu fonksiyon, “*user behavior model*”, “*makespan*” veya “*waiting times*” gibi klasik parametrelere ek olarak, kullanıcı tatminini tahmin için de bir model (user satisfaction) içerir. Sistem kullanıcı davranışını öğrenir ve kaynakların kullanımlarını tahmin edebilir.

2.4 Bulut Bilişim Mimarileri

Bulut bilişim (Cloud computing) platformları [27], sanallaştırılmış bilgisayarların kullanıcılara, kullandığın kadar öde prensibi ile kiralanmasına olanak veren büyük ölçekli dağıtık platformlardır (Şekil 2.6). Kullanıcıya sanal servisler ağı sunulur.



Şekil 2.6 : Cloud katmanları özeti [28].

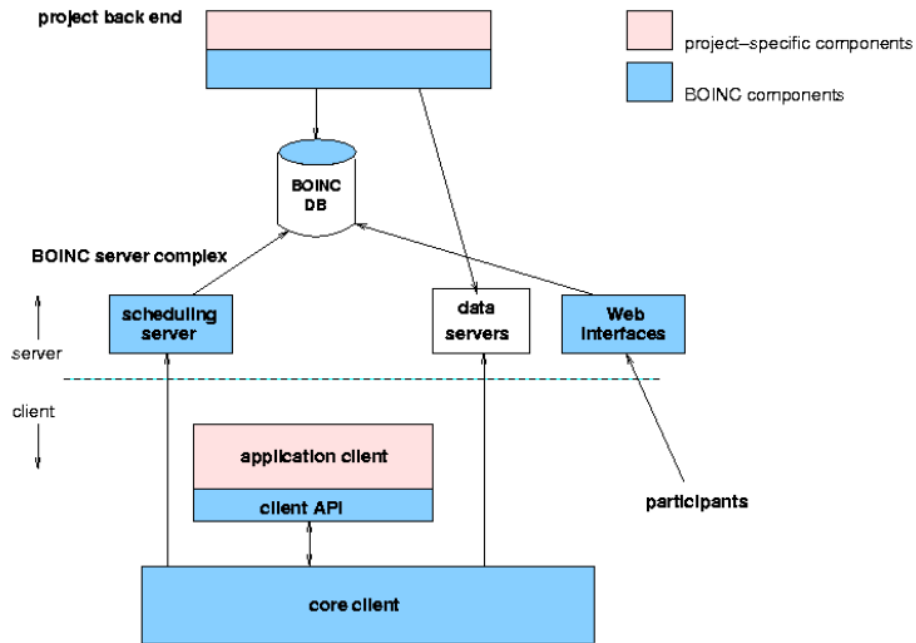
Kullanıcı, belirlenmiş QoS kriterlerine uygun olarak, herhangi bir yerden erişerek uygulamalarını bu sisteme yükleyip çalıştırabilir. Aslında YBH sistemleri de bulut bilişim platformları üzerinde çalıştırılabilen uygulamalardır. Donanımı satın almak yerine kiralamak ve kullanılan kaynakların dinamik olarak artırılıp/azaltılabilmesi, bu sistemleri cazip kılar. SLA mekanizmaları desteklenir. Bulut bilişimde verilen hizmetler literatürde aşağıdaki gibi isimlendirilir:

- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)
- Software as a Service (SaaS)

Popüler bulut mimarilerinden olan OpenStack ile ilgili daha detaylı bir inceleme Bölüm 6.2.2 'de verilmiştir.

2.5 Diğer Hesaplama Mimarıleri

Peer-to-peer ve Volunteer hesaplama mimarıleri, kişisel bilgisayarların artan performansı ve yüksek hızlı Internet bağlantıları ile mümkün olmuştur. Örnek olarak, içerik paylaşımı, bilimsel hesaplama, dağıtık veri depolama, kullanma, dosya paylaşımı, dijital çevrim-içi yayın amaçlı platformlar verilebilir. Interneti veya başka bir şebekeyi taşıyıcı (overlay network) ortam olarak kullanıp kendi mantıksal linkleri ile haberleşen özel ağlarla kurulurlar. Katılan bilgisayarların hepsi eşit statüdedir. Yönetici makine yoktur.



Şekil 2.7 : BOINC mimarisi [29].

Katılımcılar yerel kaynaklarının bir kısmını diğerlerinin kullanımına açarlar. Genellikle, TCP/IP üzerinde çalışan, DHT (Distributed Hash Tables), KAD ve Kademlia, Kazaa, Gnutella, JXTA gibi protokoller ile haberleşirler.

Volunteer hesaplama sistemleri, milyonlarca şahsi bilgisayarın geçici ve gönüllü olarak bir proje için bir araya gelmesi ile oluşur. Grid'den farkları, tüm sistemin tek bir sahibi yoktur, yazılımlar merkezi olarak yönetilmez, istemci makinalar işi sunucudan kendileri çeker ve makinaların düşük sürekliliği vardır.

Açık kaynak kodlu BOINC [29], yaklaşık bir milyon bilgisayar sayısına ulaşabilmiş, en popüler volunteer computing altyapısıdır. Paralel olarak çalışmakta olan, birbirinden bağımsız projeler bulunmaktadır (SETI, LHC, Einstein, Africa, vb.). Her proje, bir grup dağıtık uygulamadan oluşur. Uygulamalar, BOINC'i, alt işleri hesaplamak için kullanırlar. Uygulama istemcileri ise PC'ler üzerinde çalışır. İş parçaları WorkUnit olarak adlandırılır ve uygulama istemcisi, girdi dosyalar, kaynak ihtiyaçları bilgilerinden oluşur. WorkUnit'lerini dağıtıp sonuçlarını toplayan İş sıralayıcı bulunur. Aynı WorkUnit için birden çok makinede işlem kopyası çalıştırılır.

2.6 Merkeziyetsiz Sistemler

Merkeziyetsiz sistemlerin en bilineni olan Blokzincir teknolojisi, kriptopara olarak kendini göstermiş olan [30], güven için araçlara olan ihtiyacı ortadan kaldıran, yıkıcı bir teknolojidir. Blokzinciri teknolojinde kullanılan yapıtaşlarının geçmişi eskiye dayanır. 2000'lerin başında eDonkey ve BitTorrent gibi örnekleri görülmeye başlanan uçtan uca (Peer To Peer – P2P) hesaplama sistem örnekleri, Dağıtık Kayıt Defterleri (Distributed Ledger Technology – DLT) adı verilen sistemlerin gelişmesiyle sürmüştür. DLT, merkezi olarak kontrol edilmeksizin ve depolanmaksızın, eş düğümler (Peer-to-Peer Nodes) tarafından yönetilen, konsensüs ile düzenlenen, farklı konumlarda çoklanmış dijital verilerdir. Blokzinciri, işlem kayıtlarını kriptografik olarak bağlantılı bloklar şeklinde organize eden, özelleşmiş bir DLT türüdür.

Dijital verilerin sahipliğinin fiziksel dünyadaki varlıklar gibi el değiştirmesini mümkün kılmak için Blokzincir teknolojisi, veriler üzerinde üç özelliği garanti eder:

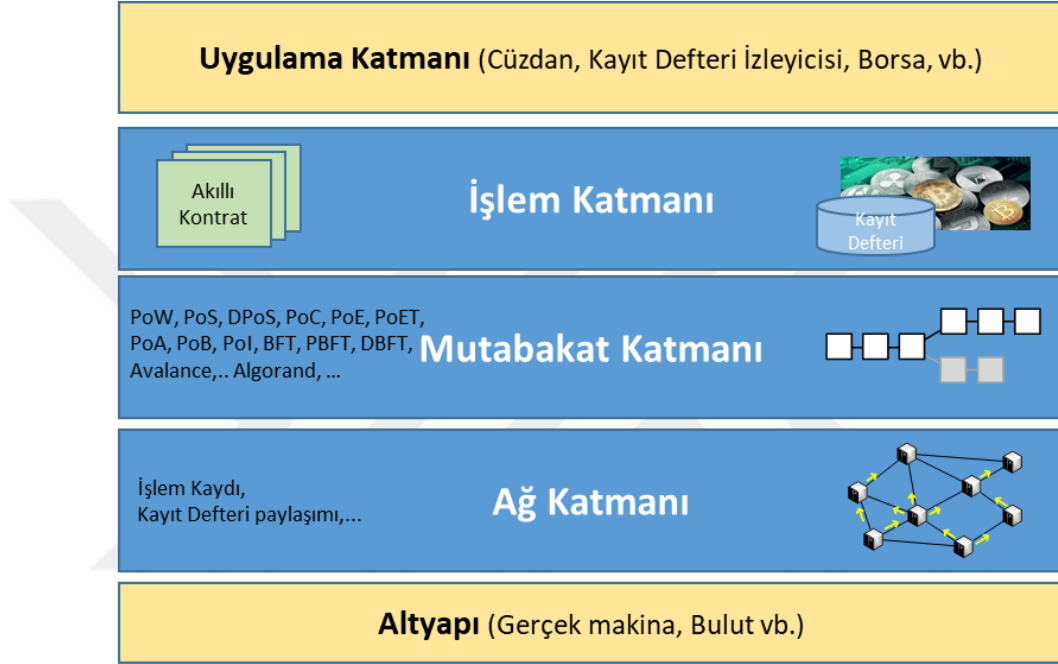
- Tekillik : Tek kopya olarak kalması gereken (para, tapu vb.) verilerin illegal kopyalarının oluşturulmasına karşı koruma.
- Doğruluk : Sadece mevcut olan digital varlıkların el değiştirilebilir olması. Kaydedilen verilerin değişmediği ve değişmeyeceği.

- Yetkinlik : El deęiřtirme iřleminin sadece yetkili kiři tarafından yapılması.

Verilerin kopyası, binlerce bilgisayarda saklanır ve veriler üzerindeki deęiřiklikler, düęümlerin mutakabatı ile yapılır. Bir veri üzerindeki her deęiřiklik, řeffaflık saęlamak üzere, zaman damgalı olarak kayıt altına alınır.

2.6.1 Blokzincir mimarisi ve yapıtařları

Bir blokzincir sistemi, mimari olarak řekil 2.8’deki katmanlarda ele alınabilir.



řekil 2.8 : Blokzincir sistemlerini oluřturan katmanlar.

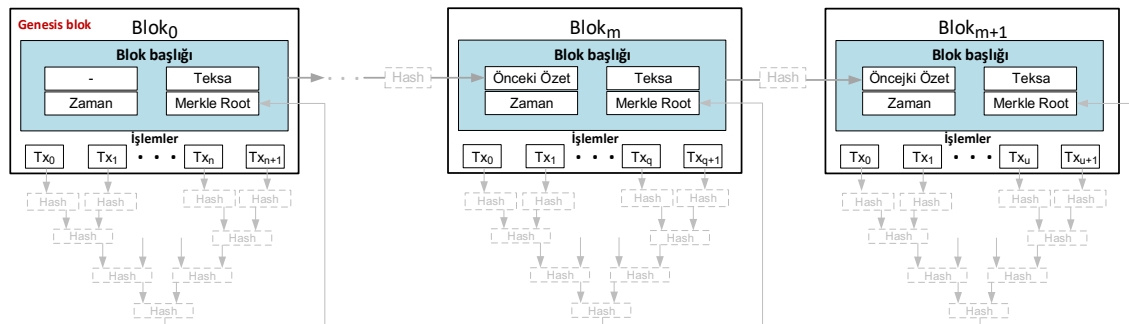
- Altyapı, blokzincirin düęümlerini oluřturan gerek veya bulut üzerinde oluřturulmuř bilgisayarlar. Mutabakat iřlemlerinden dolayı, altyapıda genellikle ok ekirdekli bilgisayarlar, GPU, FPGA ve ASIC tabanlı yksek iřlem gcne sahip donanımlar kullanılır.
- Aę Katmanı, yaygın olarak Gossip protokollerinin [31] kullanıldıęı, iřlemlerin, blokzincir düęümleri arasında hızlıca yayılması, eř düęümlerin bulunması, blokzincir verisinin indirilmesi, blokların aęda yayımlanması iřlemlerinin yerine getirildięi seviyedir.
- Mutabakat katmanı, blokzincir düęümlerinin, kayıt defterine doęru ve tutarlı veri yazılmasını garanti altına almak üzere aralarında iřlettikleri uzlařma protokolleri katmanıdır. Blokzincirin bařarımını belirleyen, kayıtların kurcalanamazlıęını saęlayan, ifte transferi engelleyen hayati bileřendir.

- İşlem/Veri (Transaction) katmanı, blokzincirinin üzerinde oluşan ve yaşayan bilgileri, akıllı kontratları ve güncellemelere ait kayıtları barındırır.
- Uygulama Katmanı, blokzincir üzerinde veri üretme, depolama ve sorgulama yapan uygulamaların bulunduğu katmandır.

2.6.1.1 Kayıtlar, işlemler ve bloklar

İşlem katmanında yönetilen, kullanıcıların blokzincire depoladıkları verileridir.

- Veriler: Blokzinciri üzerinde tutulan, uygulama alanına özgü içerik bilgileridir (kriptopara, hisse senedi, diploma, dijital kimlik, müşteri kayıtları vb. değerli bilgiler olabilir). Kayıtların blokzincir üzerinde depolanması için iki model vardır. *UTXO (Unspent Transaction Output)* modeli, işlem kayıtlarının (transaction) içine kriptoparaların kaydedilmesine ve bu işlemlerin başka işlemlerin girdisi olarak kullanılabilmesine (harcanmasına) olanak sağlar. Blokzincirde yönetilen dijital varlıklar, işlem kayıtları içinde dağılmış olarak tutulur. *Account* modelinde ise işlem geçmişine ek olarak işlemler oluştuğunda kümülatif olarak güncellenen bir hesap bakiye tablosu (account) tutulur. Bu modelde ayrıca akıllı kontratlar ve onların kendi iç verileri (state) tutulur.
- İşlemler (Transaction): Blokzincir üzerinde işlenen verilerin (asset) oluşturulması ve güncellenmesi ve yok edilmesi amaçlı atomik isteklerdir (para transferi, akıllı kontrat işlemleri, yetki devirleri vb.)
- Bloklar: İşlem kayıtları gruplanarak, blok adı verilen veri yapıları içine yazılır. Blok başlığına yazılacak özet (hash) hesaplanırken bir önceki bloğun özet değeri de girdi olarak eklendiği için bloklar geriye doğru birbirlerine kriptografik bağ ile bağlıdır (Şekil 2.9).



Şekil 2.9 : Blokları arası Özet Zinciri yapısı (Bitcoin).

2.6.1.2 Blokzincir sisteminin aktörleri

Blokzinciri sistemlerinde aşağıdaki tablodaki aktörler bulunur (Çizelge 2.1). Sistemin güvenliği, bu aktörler sistemden farklı açılardan elde ettikleri çıkarlarını korumak ve maksimize etmek amacıyla hareket ederken otomatik olarak sağlanmaktadır.

Çizelge 2.1 : Blokzincir platform aktörleri ve sistemden beklentileri.

Aktör	Rolü	Beklentisi (motivasyonu)
Madenciler (Kriptopara Blokzincirlerinde)	Blokzincirin p2p şebekesini oluşturan düğüm bilgisayar(lar)ını işletir	- Blok üretiminden dolayı sistemden ödül para kazanmak - Blok içine yerleştirilen işlemlerin sahiplerinden alınan komisyon ücretleri
Onay Düğümü sahipleri (Konsorsiyum Blokzincirlerinde)	Blokzincirin p2p şebekesini oluşturan düğüm bilgisayar(lar)ını işletir	İş Modeli verilerinin Blokzincir üzerinde güvenli olarak yaşaması sonucu sağlanacak fayda (güvenlik, hız, maliyet kazanımları)
Geliştiriciler	Platformu geliştirmek ve/veya sorunlarını çözmeye, yetenekler eklemeye çalışmak	- Sistem kurulurken boş olmayan cüzdan - Token satışından kazanılan para - Ün - Yönetişimde rol alma
Kullanıcılar	Blokzinciri, dijital veri depolamak için kullananlar. Kendi iş süreçlerini blokzincir üzerinde çalışacak şekilde kurgulayıp işletirler.	- Verilerin güvenli tutulması - Otoriteden bağımsızlık (Kripto para ise) değer kazanması, 7/24 kolay transferi - Aracısız, kesintisiz, şeffaf işlem
Borsa sahipleri (Kriptopara Blokzincirlerinde) ve Yatırımcılar	Blokzincir üzerindeki (değerli) dijital verilerin, gerçek dünya para sistemleri ile alınıp satılmasına aracılık ederler.	- Alım satım yapan kullanıcıların ödediği komisyonlar - Yatırım yapılan Blokzincir projesinin değerlendirilmesi ile elde edilen kazanç

2.6.1.3 Kriptografik yapıtaşları

Kullanılan kriptografik bileşenler, kullanım amaçlarına göre ikiye ayrılır:

- Blokzincir sisteminin kendi güvenliğini sağlayanlar (İmza, özet)
- Kullanıcıların verileri için mahremiyet ve anonimlik sağlama amaçlı, gelişmiş algoritmalar/protokoller (özel imzalama, sıfır bilgi ispat protokolleri, akümülatörler, homomorfik şifreleme, çok taraflı kriptografi vb.)

2.6.1.4 Mutabakat protokolleri

Blokszinciri platformunun karakteristiğini oluşturan ve kayıtların bütün düğümlerdeki kopyalarının aynı olmasını sağlayan bileşendir. Bilinen yetmişten fazla mutabakat protokolü vardır [30]. Kullanılan mutabakat protokolü, blokszincire erişim politikasını, sağlayacağı hızı ve güven seviyesini belirler. Bir Açık Blokszincirde risk daha yüksek olduğu için PoW (Proof of Work) gibi daha güçlü (ama yavaş ve maliyetli) algoritmalar kullanılır. PoW tabanlı mutabakat, ilk olarak Bitcoin ile kendisini ispatladığı için Nakamoto konsensüs yöntemi olarak da adlandırılır [32]. Yöntemin güvenliği, blokszincirin toplam işlem gücünün %51'den fazlasını dürüst madencilerin kontrolünde bulunması varsayımına dayalıdır. Bu yöntemde, sistemin kayıt defterine yeni bir blok üretip eklemek isteyen blokszincir düğümlerinin (madenciler), sistemin belirlediği zorluk derecesindeki bir kriptografik bulmacayı çözmesi gerekir. Bu bulmacanın çözülmesi ile (blok özet değerinin belirli bir değerden küçük olmasını sağlayan bir nonce bulunmasıyla) blok, kayıt zincirine eklenmeye hazır hale gelir. Madencilik (mining) ismi ile bilinen bu yöntemle sistemde para üretimi de gerçekleşmiş olur. Üretilen bu blok, diğer düğümlere de bildirilir. Diğer madenciler, aldıkları bu blok duyurusunu, bazı kontrollerden geçirerek (kayıt defterinin geçmişi ile uyum, vb.) kendi kayıt defterlerine eklerler. Mutabakat protokolü gereği, madenciler, kendilerine ulaşan blok duyurularına bakarak, varsa alternatif zincir dalları arasında, en uzun olan dalı blokszincirini asıl zincir olarak kabul ederek kendi verilerini güncellerler (Longest chain wins).

Bir diğer protokol olan PoS (Proof Of Steak), üretilen blokların doğrulaması ve onaylanmasında, madencilere sistemdeki kriptopara hisseleri ile orantılı söz hakkı verir. Hile yaptığı tesbit edilen madencinin hisseleri elinden alınır.

İzinli blokszinciri türlerinde ise paydaşların bilinir olması ve düğümlerin çalışma ortamının daha kontrollü olması nedeniyle genelde, düğümler arası etkileşim (oylama vb.) gerektiren ve madencilerin 1/3'ünden azının kötü niyetli olduğu varsayımına dayanan BFT (Byzantine Fault Tolerance) isimli mutabakat yönetiminin [33] türevleri kullanılır. Çoklu mutabakat algoritmasını içeren hibrit örnekler de vardır.

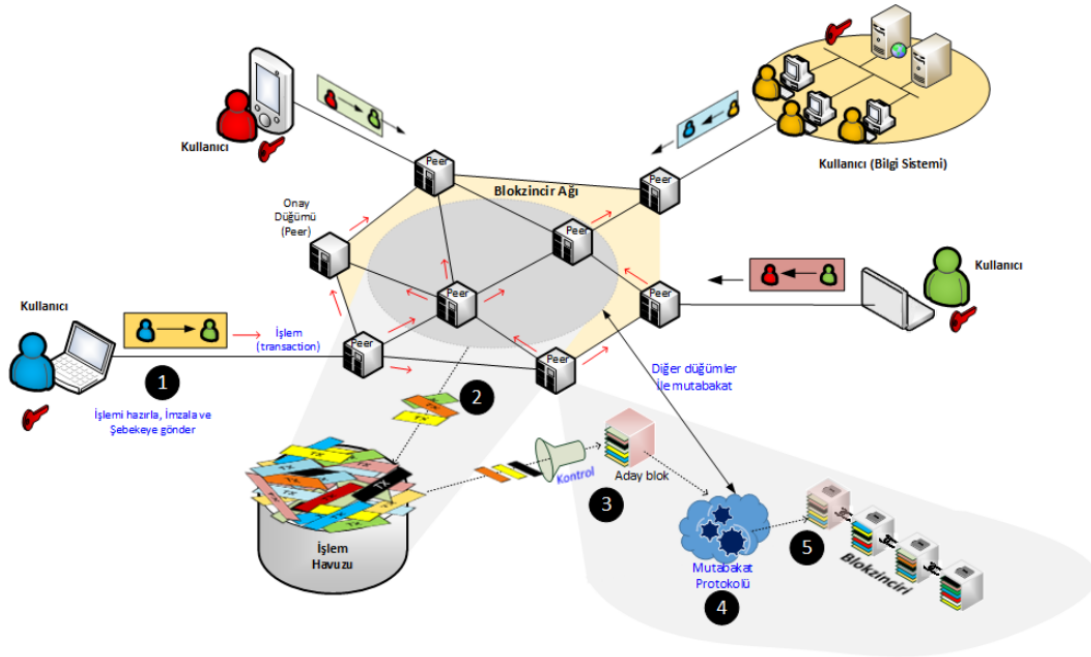
2.6.1.5 Akıllı kontratlar

Akıllı kontrat (Smart Contract) fikrini 1994 yılında ilk ortaya atan, Nick Szabo'dur. İçlerinde, blokszincirin güvenlik yapısı ile koruma altına alınmış veriler depolayabilen,

kopyası her bir blokzincir düğümü üzerinde çalışan, izinsiz olarak durdurulamayan ve değiştirilemeyen kodlardır. Bunlar, güvenilir üçüncü taraflara (merkezi otoritelere) ihtiyaç duyulmadan, paydaşları arasındaki güveni tesis ederek bir iş mantığının şeffaf izlenebilir şekilde gerçekleşip yürütülmesini sağlarlar. Bu sayede, blokzincirinin pek çok sektörde kullanımının önü açılmıştır. Akıllı kontratların kendi hesap adresleri vardır ve içlerinde kriptopara tutabilirler. Diğer akıllı kontratlar ve kullanıcı hesapları ile etkileşebilirler. Ethereum, Cardano, Hyperledger Fabric, Corda, Quorum gibi, akıllı kontrat destekleyen pek çok blokzincir platformu geliştirilmiştir.

2.6.2 Blokzincirin işleyişi

Blokzincir sisteminde, olaylar, mevcut veriler üzerinde bir güncelleme isteğinin (para aktarma, bir dijital verinin sahipliğini devretme, ortak veriyi güncelleme vb.) kullanıcı tarafından İşlem (*Transaction*) olarak hazırlanıp, blokzincir ağının düğümlerinden birisine göndermesi ile başlar. Şekil 2.10'da bu detay gösterilmiştir.



Şekil 2.10 : Blokzincir sisteminde veri güncelleme akışı.

Kullanıcı bu işlem isteğinin içine, sistemin iş mantığının gerektirdiği kriptografik bileşenleri de (imza, bazı kanıtlar vb.) koymak zorundadır. Blokzincir ağına aynı anda pek çok kullanıcı, farklı düğümler üzerinden İşlem isteği gönderir. Bu istekleri alan Blokzincir düğümleri, istekleri, kendi komşu düğümlerine de yayarlar. İşlemler, bazı kontrollerden (gönderenin imzası, kayıt defteri içeriği ile uyumu vb.) geçtikten sonra, sistem genelindeki her düğümde kopyası olan sanal bir İşlem Havuzu'nda toplanır.

Düğüm, işlem havuzundaki İşlemleri, diğer düğümler ile işbirliği içinde (mutabakat protokolleri sayesinde) işleyip bloklara çevirerek Kayıt Defterinin (Blokzinciri), bütün düğümlerde bulunan kopyaları aynı olacak şekilde güncellerler.

2.6.3 Blokzincir platformları

Blokzincir platformları, blokzincir verilerine kimin erişebildiğine ve blokzincir bilgisayarlarını kimin işlettiğine bağlı olarak alt sınıflara bölünür. Açık (Public, permissionless) blokzincir ağına isteyen herkes katılabilir. Bitcoin ve Ethereum bu türün en çok bilinen örnekleridir. Özel (Private) blokzincirlerinde ise ağa yeni düğümlerin katılımı, ağın kurucularının daveti ve/veya tanımladıkları kurallara göre olabilir. İzinli (permissioned) blokzincir ise Açık ve Özel tipler arasındaki bir hibrit türdür. Önceden belirlenmiş sınırlı sayıda düğüm bulunur. Konsorsiyum veya federe blokzincir adı da verilir (Örnek Ripple kripto para sistemi).

Bitcoin en çok bilinen kriptopara odaklı blokzincir platformudur. Bitcoin ağındaki madenci bilgisayarlarının toplamı (Mart 2022 tarihinde), saniyede yaklaşık 200 milyon terahash işlem kapasitesindedir. Ethereum, kriptopara olarak kullanılmanın yanında, uygulama geliştiricilerin, kendi merkezi olmayan uygulamalarını geliştirmesine ve çalıştırmasına olanak sağlayan, akıllı kontrat desteği sağlayan bir platformdur. “Kurumsal Odaklı Ethereum” olarak nitelendirilen Quorum adlı platform, düğümler üzerindeki kriptografik bileşenler sayesinde, içeriği gizlenebilen işlemler yapılmasına da olanak sağlar.

Hyperledger, Linux Vakfı tarafından Aralık 2015’te başlatılan açık kaynak kodlu bir blokzincir projeleri platformudur. Hyperledger kapsamındaki projelerin en bilinenlerinden olan Hyperledger Fabric blokzincir platformu, kurumsal dönüşüm projelerini hedefleyen izinli bir blokzinciri platformudur. Hyperledger Indy blokzincir platformu ise merkezi olmayan dijital kimlik yönetimi konusunda özelleşmiş, izinli blokzincir türüdür.

Pek çok özel amaçlı blokzincirlerinin yanında (Ripple vb.), bir konsorsiyum tarafından geliştirilen Corda platformu, işletmeler arasında yasal sözleşmeleri kaydetmek, yönetmek ve otomatikleştirmek ve finansal piyasalardaki uygulamalara çözümler sunmak için tasarlanmış genel amaçlı bir blokzinciri platformudur

3. POLİTİKA TABANLI YÖNETİM VE POLICE PBM ALTYAPISI

Bu bölümde, Politika Tabanlı Yönetim (Policy Based Management-PBM) kavramları üzerinde yürütülen literatür çalışmasının sonuçları özetlenmiştir. PBM yapıları, uygulama mimarileri, politika dilleri ve geliştirilen araçlar tanıtılmıştır. Tez hazırlık çalışmasında geliştirilerek tez kapsamında YBH sistem yönetimi için kullanılmak üzere genişletilen Police PBM platformu da bu bölümde tanıtılmıştır.

3.1 Politika Tabanlı Yönetim (PBM)

IT sistemlerinin yönetimi, genelde sistemin içerdiği cihazlar tek tek manuel konfigüre edilerek ve sonrasında bu cihazlar hata olasılığına karşı izlenerek yapılır. Bu işlemleri yapmada kullanılan yazılım araçlarından bir veya fazla ile OSI'nin FCAPS (Fault, Configuration, Asset, Performance, Security) [1], [2] adlı yönetim hedefleri karşılanmaya çalışılır. Ama yönetilen cihazlar gittikçe karmaşıklaştığı, sayısı ve aralarında bağımlılıkları arttığı için, manuel ayar yaparak yönetim gittikçe zorlaşmaktadır. PBM, bu zorlukları aşmak için ortaya çıkmış bir yönetim yaklaşımıdır. Çeşitli ticari (HP-OpenView [34], IBM-NetView [35], CA-UniCenterTNG, Sun-NetManager, vb.) ve akademik PBM sistemleri [36-39] mevcuttur.

PBM sayesinde, yönetilen sistemin davranışları, tanımlanan yüksek düzeyli hedeflere (politika) göre otomatik olarak şekillendiği için operatörlerin yönetim yükü azalır. PBM sistemi, politikaları, yönetilen cihazların anlayabileceği komut ve konfigürasyonlara otomatik olarak dönüştürdüğü için, operatörlerin yönetilen cihazların teknik detaylarını bilmesine gerek kalmaz. Cihazlar teker teker elle konfigüre etmeye göre daha tutarlı, hızlı ve az hatayla yönetim mümkün olur. PBM kavramı ile sağlanan ana faydalar şunlardır:

- Sistemler geleneksel yönetim araçlarına göre daha üst seviyeden yönetilebilir,
- Cihazların teknik detaylarını öğrenme yükünden kurtarır,
- Cihazların konfigürasyonu, operatörlere saydam ve otomatik yapılır,
- Sistemleri kesintiye uğratmadan davranışlarını değiştirebilir.
- Sistem üzerinde daha iyi kontrol sağlar,

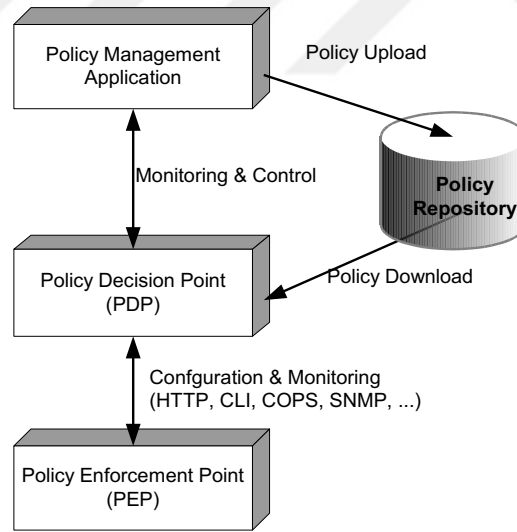
- Proaktif yönetimi kolaylaştırır,
- Uzman kişilerin bilgileri politikalar olarak yeniden kullanılabilir,
- Problem durumlarını insan müdahalesi olmadan da düzeltebilir.

PBM genellikle, Güvenlik, QoS ve ağ trafiğini yönlendirme yönetimi [40-43] gibi amaçlarla kullanılır. Güvenlik uzayında ise [44] genellikle Giriş Denetimi (Admission Control) ve Erişim Denetimi (Access Control) amacıyla kullanılır.

YBH yönetimi, PBM için yeni bir uygulama alanı olabilir. YBH yönetimi, daha çok sayıda cihazın daha karmaşık şekilde konfigüre edilmesi ile yapılır. Yönetimi basitleştirmek, operatörlerin yükünü hafifletmek, iş sıralama yönetimi ve kaynak paylaşım yönetimi için PBM sistemleri kullanılabilir.

3.1.1 Genel PBM mimarisi

IETF, PBM için yaygın kabul gören bir bilgi modeli önermiştir [45, 46]. Bu modele göre tipik bir PBM mimarisi (Şekil 3.1), bir politika yönetim uygulaması (PMA), bir politika deposu, politika karar noktası (PDP) ve bir dizi politika uygulama noktası (PEP) içerir. PEP, yönetilen kaynaklar üzerinde veya onlara yakın konuşlu etmendir.



Şekil 3.1 : Genel PBM mimarisi [46].

Politikalar, üst düzey amaçları belirten ifadelerdir. Tanımlanan politikalar, politika deposunda saklanır. PDP, politika deposundaki politikaları değerlendirip gerekiyorsa PEP'lerin anlayacağı formata çevirdikten sonra uygulanmak üzere PEP'lere gönderir. PEP'ler, yönetilen sistemde oluşan değişiklikleri PDP'ye bildirerek, güncel politikalara göre atılacak adımı sorgularlar. PEP'ler, kendilerine gelen politikalara veya PDP'den aldıkları cevaplara göre davranışlarını şekillendirirler.

Politikalar PBM sistem elemanları arasında, IETF-COPS (Common Open Policy Service), SNMP, CLI, Telnet, JMS, LDAP, ODBC, RMI, CORBA [47], XML, IETF SPP (Security Policy Protocol) [48] ve COPS (Common Open Policy Service) [49] gibi protokoller ile taşınır. Politika depoları genelde LDAP (Lightweight Directory Access Protocol) [50] dizin servisleri olarak gerçekleştirilir.

3.1.2 PBM ürünleri için gereksinimler

PBM alanındaki çalışmaların çoğu, politikaların nasıl depolanıp uygulanacağı konularına yoğunlaşmıştır. Politika tanımlama dili olarak pek çok alternatif karmaşıklıkta dil önerilmiştir [14]. IETF tarafından önerilen if-koşul-then-eylem formatındaki ilkel dil [45], en yaygın kullanılan dildir. Literatürden [51, 52] derlediğimiz aşağıdaki liste, bir PBM yazılımından beklenen özellikleri içerir:

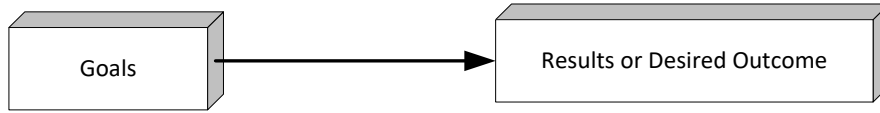
- Politika dili, insanlar tarafından anlaşılabilir ve bilgisayarlar tarafından uygulanabilir karmaşıklıkta olmalıdır.
- Emir ve Yetki tipinden politikalar tanımlanabilmelidir.
- Yönetilen sistemdeki olaylara cevap olan eylemler tanımlanabilmelidir.
- Pozitif ve negatif politikalar tanımlanabilmelidir.
- Delegasyon politikaları ile yetki devrine olanak sağlamalıdır.
- Periyodik ve periyodik olmayan eylemlerin yürütülmesi tanımlanabilmelidir.
- Politikalar, yönetilen sistemin teknik detaylarından bağımsız olmalıdır. Farklı üreticilerin sistemlerinin yönetiminde kullanılabilir olmalıdır.
- Politikaların uygulanması, sistemdeki çeşitli koşullara göre sınırlandırılabilir (Kısıt ifadeleri ile).
- Tek nesneler için ve nesne grupları için politikalar tanımlanabilmelidir.
- Sistem genişletilebilir ve ölçeklenebilir olmalıdır.
- Tanımlanmış politikaların tekrar kullanılabilirliği desteklemelidir.
- PBM bileşenlerinin kendisi de yönetilebilir (Self-management).
- Politikalar arasındaki çakışmaların tesbit ve çözümüne destek sağlamalıdır.

3.1.3 Politikalar ve tanımlama dilleri

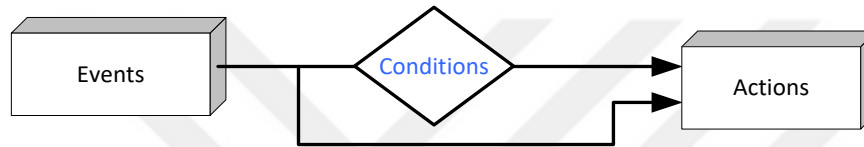
Politika kavramının, akademik ve ticari birçok tanımı bulunmaktadır (Şekil 3.2):

- Şimdiki ya da gelecekteki kararları belirlemek ve yönlendirmek için gerekli olan kesin bir amaç veya eylem ifadesi [RFC3198].

- Politikalar ağ kaynaklarını yönetmek ve erişimlerini denetlemek için kullanılan kurallar bütünüdür. [RFC3060].
- Yönetim amaçlarından türetilmiş, bir sistemin davranışındaki seçimleri belirleyen kural tanımlarıdır [53, 54].



If Administrator password entered and validated using appropriate shared secret, **then** assume that trust is established



If FileSystemThreshold event generated, (AND Unused storage is > 1G), **then** allocate 100MB of storage and add this to the FileSystem's StorageVolume.

Şekil 3.2 : Politika tanımlama tipleri [53].

Politika dillerinin sınıflandırması için literatürde çeşitli öneriler olsa da ([53], [55], [56]) en çok kabul gören politika tanımlama seviyeleri aşağıdakilerdir [51]:

- Üst-seviye soyut (High-level) politikalar: Doğal dilde cümlelerdir. Uygulanmak için önce daha alt seviyedeki politikalara dönüştürülürler.
- Alt-seviye politikalar: Cihaz konfigürasyonları, güvenlik duvarı filtreleri vb.
- Tanımlama-seviyesi politikalar (ağ-seviyesi politikalar): Özel bir formatta tanımlanırlar ve yorumlanıp yürütülmesi otomatize edilebilir. Formal mantık-tabanlı, Kural-tabanlı ve Özel dil alt sınıfları vardır. Modelimizi tasarlarken ilham aldığımız dil bu tiptendir.

3.1.3.1 Mantık-tabanlı diller

Formal mantık-tabanlı politika dillerini operatörlerin okuyup yorumlaması kolay değildir. Formal Hesaplama yöntemlerinden olan, Prolog ve diğer teorem ispatlayıcı araçları kullanabilme olanağı sağlarlar. ASL (Authorisation Specification Language) [57] erişim denetim için kullanılan bir örnektir. Örnek iki politika aşağıda verilmiştir:

```
// ULK grubunda olup TDR grubunda olmayan bütün aktörler,
cluster1 üzerinde iş yükü çalıştırmaya yetkilidir.
cando(cluster1, s, +runjob)  $\leftarrow$  in(s, ULK) &  $\neg$ in(s, TDR)
// Admin3 grubunun bütün üyelerinden, cluster1 üzerindeki
işleri durdurma yetkisi alınmıştır.
cando(cluster1, s, -stopjob)  $\leftarrow$  in(s, Admin3)
```

[58]'de tanıtilen formal dil ise yetki ve emir/yükümlülük (obligation) politikaları tanımlanabilen bir dildir. Bu dilde kullanılan aksiyom; $Pp = \neg O\neg p$ ("permitted p is equivalent to not p being not obliged"). Anlamı, eğer bir eylem için aktöre yetki verilmiş ise aksini yaptıracak bir emir politikası bulunamaz. Yani, bir emir politikasının eylemini yürütmek için, ayrıca bir yetki politikası olması gerekmez. Bu yaklaşımın aksini savunan Ponder [36] gibi diller de vardır (Yani, emir politikasının uygulanabilir olması için aynı eyleme ait bir yetki politikası da bulunması gerekir).

3.1.3.2 Kural-tabanlı diller

PDL (Policy Description Language) [59], en bilinen olaylarla tetiklenen kural tabanlı dillerdendir. PDL, Bölüm 3.2.2.3'te anlatıldığı gibi basit olaylardan birleşik olaylar oluşturmayı da destekler. Söz dizimi aşağıda verilmiştir:

Politika kuralları:

event **causes** action **if** condition

//condition geçerli iken olay oluşursa eylem yürütülür.

Politika tanımlı olay ifadesi:

event **triggers** policy-defined-event **if** condition

// condition geçerli iken olay oluşursa, olay tetiklenir.

En çok kullanılan IETF Politika dili ise aşağıdaki formatta kurallar içeren dildir.

if <condition(s)> **then** <action(s)>

Ağ trafiği yönetimi amaçlı olarak geliştirilen PPL (Path based Policy Language) [14, 60] dili, IETF'nin if-koşul-then-eylem formatındaki modelini [61] genişletir.

3.1.3.3 Özel tanımlama diller

Genelde nesneye dayalıdır ve yönetilen sistem elemanları nesneler olarak, nesnelerin metodları ise yürütülebilecek eylemler olarak modellenir. Bu nesnelerden bir kısmı aktör (subject), diğerleri ise aktörlerce metodları çağrılan hedef nesne (target) olarak isimlendirilir. SPL (Security Policy Language) [62], olay tabanlı bir dildir (Bir aktörün

bir hedef nesneye erişme isteği olay olarak modellenir). Ponder dili, Londra Imperial College’de geliştirilen ve en çok bilinen dillerdendir [36, 63-65]. Ponder, literatürdeki pek çok çalışmadan (PDL[59], [66], SPL[62], Tower[67], vb.) aldığı fikirleri birleştiren bir dildir [63], [65], [68].

Ponder dili, Emir (*Obligation*), Yetki (*Authorisation*), Delegasyon (*Delegation*), ve Sakınma/Kaçınma (*Refrain*) tipinden politikaları destekler (Bkz. Şekil 3.3). *Yetki* politikaları aktörün belirli hedef nesne(ler) üzerinde izinli ve yasaklı olduğu aktiviteleri tanımlar. *Emir* politikaları ise olaylarla tetiklenen *condition-action* tipli kurallardır ve bir aktörün (yönetim sorumluluğundaki) belirli hedef nesne(ler) üzerinde hangi eylemleri yapması veya yapmaması gerektiğini tanımlar. *Sakınma* (*Refrain*) politikaları ise aslında negatif emir politikalarıdır.

<pre>//Authorisation policy inst (auth+ auth-) politikaName { subject [<type>] dSE; target [<type>] dSE; action action-list; [when constraintExpression;] }</pre>	<pre>inst auth+ sepDuty { subject s= accountants; action approvePayment; target t= cheques when s.id <> t.issuerID }</pre>
<pre>//Obligation policy inst oblig policyName { on event-specification; subject [<type>] dSE; [target [<type>] dSE;] do obligationActionList; [catch exceptionSpecification;] [when constraintExpression;] }</pre>	<pre>inst oblig LoginFailure { on 3*loginfail(userid); subject s= /Student/Admin target t= /Student/users ^{userid} do t.disable()->s.log(userid) }</pre>
<pre>//Refrain policy inst refrain policyName { subject [<type>] dSE; target [<type>] dSE; action actionList; [when constraintExpression;] }</pre>	<pre>inst refrain testingRes { subject s=/test-engineers; action DiscloseTestResults(); target /analysts+/developers when s.testing_sequence = "in-progress" }</pre>
<pre>//Delegation policy inst deleg+ (associatedPolicyName) policyName { grantee [<type>] dSE; [subject [<type>] dSE;] [target [<type>] dSE;] [action action-list ;] [when constraintExpression ;] [valid constraintExpression ;] [hops int-value ;] }</pre>	<pre>inst deleg+ (switchProfileOps) delegSwitchOps { grantee /DomainAdmin ; target /switches/typeA ; action enable(), disable(); valid Time.duration(24) ; }</pre>
<pre>//instance syntax type (auth+ auth-) policyType (formalParameters) { subject [<type>] dSE; target [<type>] dSE; action action-list ; [when constraint-Expression ;] } inst (auth+ auth-) policyName = policyType(actualParameters);</pre>	

Şekil 3.3 : Ponder politika tanımlama dili örnekleri.

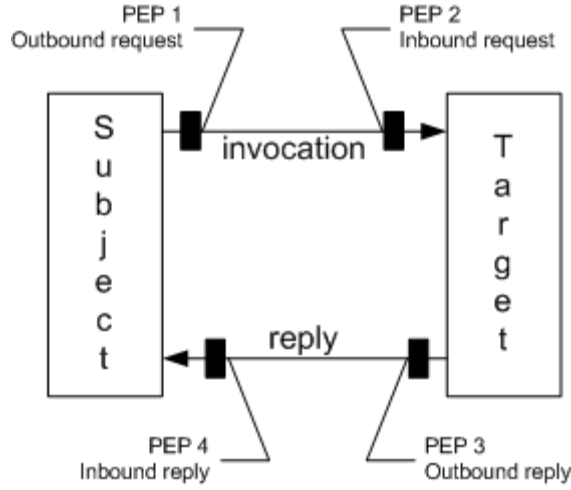
Meta Politikalar ise politikalar arası bağımlılıkları tanımlayan politiklardır. Meta politikalar, merkezi olarak, politikadaki bazı çakışmalar çözmeye çalışılır [69].

3.1.4 PBM ürünleri

Bu bölüm, genel PBM amacıyla kullanılabilecek olan akademik ve ticari ürünler hakkında bilgi verir. Ticari ürünlerin çoğu, mevcut halleri ile YBH mimarileri için doğrudan kullanılabilir değildir. YBH mimarilerinin, erişim denetimi, güvenlik, vb. ihtiyaçlarından bazılarını karşılamak üzere entegre edilebilirler. Bölümün devamında ise YBH sistemleri için geliştirilmeye çalışılan PBM ürünlerinden bahsedilmiştir.

Çoğu ürün IETF modeline uygun geliştirilmiştir. [70] çalışması, IETF Modeline göre ve sadece IPSec filtrelerini tanımlamak üzere geliştirilmiştir. [70] ve [71] ürünleri, diğer çoğu ürünlerde olduğu gibi, özel bir uygulama alanına (güvenlik alanı) özgü ürünlerdir. [72] ise QoS ve SLA yönetimi konusunda geliştirilmiş bir üründür.

Ponder modeli [65], pek çok çalışmaya ilham vermiştir. Politikaların yürütülmesi için gerekli altyapıyı sağlayan, kaynaklar, kullanıcılar ve politika bilgilerinin hepsini nesneler olarak modeller. İki tip politika uygulama etmeni kullanır; yetki politikaları için her hedef (target) nesnenin *Erişim Denetçisine (AC)*, sakınma ve emir politikaları için ise her aktörün *Politika Yönetim Etmenine (PMA)* işleri yaptırırlar. Politikaları yürütmek için, politika ile birlikte yönetilen sistemin teknolojisine uyan bir yürütme sınıfı da üretir (enforcement class). *Police* modeli ise bir yürütme sınıfı yerine, yönetilen platformdan bağımsız yapmak için yönetilen sistemin detaylarını politika yürütme sürecinden gizleyen bir uyarılma katmanı (Bölüm 3.2.3.4) içerir. Öte yandan *Ponder2* ise *Ponder*'ın daha sonra geliştirilen sürümüdür (Şekil 3.4). Bu sürüm, merkezi ve çevrim-dışı politika tanımlanması (yönetilen sistem ile etkileşimsizlik), politika uygulama bileşenlerinin birbiri ile etkileşim eksikliği, küçük cihazlarda çalıştırılabilir olmamaları gibi eksiklikleri çözmek için geliştirilmiştir. En önemli değişikliklerden birisi, Police modelimizdeki gibi, aktörlerin eylem girişimini daha baştan kontrol altına alma değişikliğidir. *Ponder2*'de, PAF (Ponder Authorisation Framework) adlı yeni kavram sayesinde kontrol noktası sayısı/çözünürlüğü artırılmıştır. PAF yetki politikaları, hem aktörü hem de hedefi koruyacak şekilde uygulanır. PAF ile 4 farklı PEP sağlanır: PEP1 ve PEP4 politikayı aktör tarafında uygular, PEP2 ve PEP3 ise hedef tarafında uygular. Politikalar, (birlikte çalışabilirlik için) artık XML formatındadır. Tanımlanınca hemen aktif hale geçerler.



Şekil 3.4 : Ponder2 çoklu PEP tasarımı.

QoS politikaları için IETF modeline uyumlu olarak geliştirilen diğer bir yönetim sistemi [73] ise Ponder dilinde tanımlanmış politikaları uzman sistem kurallarına dönüştürür. Her yönetim uzayı için ayrı bir PDP, politikaları alır ve işler. Bir uygulama İsim Sunucusuna kayıt olduktan sonra, QoS isteklerini PDP'ye iletir. PDP, Ponder dilinde yazılmış politikaları, JESS (Java Expert System Shell) kurallarına çevirir.

[74] modeli ise, yönetilen elemanları herhangi bir ön bilgi olmaksızın yürütme anında keşfetmek için CIMOM tabanlı bir WBEM (Web based Enterprise Management) [75] yapısıdır. Bu yaklaşım da Police modelimizde kullanılmıştır.

BANDS [76] isimli ürün ise etki alanları arasında politika pazarlığı ve üretimi için dağıtık bir çatı ve dual dil içerir. Alt seviye politika üretimi, üst seviye politikaların rafine edilmesi ile değil, yorumlanması ile gerçekleştirilir. Gereksinimleri ifade etmek için *SRL (Security Requirement Specification Language)* dili kullanılır. Tanımlanan, bu gereksinimler üzerinden, tüm gereksinimleri karşılayacak IPsec güvenlik politika kuralları otomatik olarak üretilir.

Verma, IETF çatısı altında politika doğrulama, dönüşüm algoritmaları, dağıtım mekanizmaları ve PEP algoritmalarının detaylı bir tanımını yapmıştır [72]. IETF modelinde politika depolama ve transfer için LDAP dizinleri önerilse de örneğin SNMPConf Çalışma Gurubu, SNMP tabanlı bir mekanizma ile politikaları transfer ederken [77], Martinez [78] Script MIB yapısını kullanır. ScriptMIB, hem politika nesnelerini hem de politika kodlarını içeren yönetim kodlarını (script), PDP olarak çalışan Script MIB etmenlerine dağıtır.

Politikaları uygulamak için ağdaki etmenlere kod parçası olarak nesneler gönderme konusunda da birçok çalışma vardır. Aktif ağ kavramını kullanmak bunlardan birisidir [79, 80]. Bir diğeri ise PDP ve PEP'lere SNMP Script MIB kullanarak kod yüklemektir [78]. Literatürde, Gezgin Etmenlerin (Mobile Agent) politika uygulanmasında kullanılması konusunda da öneriler vardır.

Aktif Ağ (Active Networking) [79] kavramı kullanarak ISP (Internet Service Provider) politikaları için esnek bir dağıtım modeli örneği [80]'de açıklanmıştır. Servis pazarlığı ve ağ cihazlarının konfigürasyonu için aktif ağ kapsülleri (active packets) kullanılır.

Bir başka akademik model [81], güvenilmeyen yazılım modüllerinin davranışlarının izlenmesi amacıyla e-ticaret uygulamasında güvenlik politikaları kullanır.

Positive PBM çatısı [82] ise ağdaki sistem ve uygulamaların güvenlik yönetimi için geliştirilmiştir. Bu çatı, CIM DMTF tabanlıdır ve iki politika dili kullanır; dillerden birisi politikaları diğeri ise yönetilen sistemin modelinin tanımlanması için kullanılır.

Ticari PBM ürünlerinin çoğu IETF modeline göre geliştirilmiştir ve uygulama alanlarına özelleşmiştir. PBM sayesinde, yönetim sistemleri, proaktif olarak QoS ve SLA yönetimi yapan sistemlere dönüşmektedir. QoS ve güvenlik yönetimi konusunda PBM ürünleri geliştiren [83, 84] firma ürünlerinin hemen hepsi, IETF politika formatını kullanır. Politikalar ise genelde LDAP dizinlerinde saklanır.

Çoğu üretici, PBM teknolojisini, şebeke cihazları yönetmek için kullansa da bilgisayarlardaki yazılımları yönetmek için kullanılan ürünler de vardır. Bulut hesaplama (cloud computing) ile birlikte, paylaşılan kaynak havuzundaki kullanıcı davranışlarının yönetilmesi de diğeri bir hedef alan olmaya başlamıştır.

Önemli PBM ürünleri listesinde, Optivity Policy Services (Nortel), Orchestream, Openview PolicyXpert (HP) [34], Policy Networking (Cisco), Policy Manager (Cisco), NetPolicy (Allot Communications), Open Policy System (IPHighway), RealNet Rules (Lucent), Visual Policy Management (SolSoft), ZENworks (Novell), eTrust (Computer Associates) ve Tivoli Management Framework (IBM) bulunur.

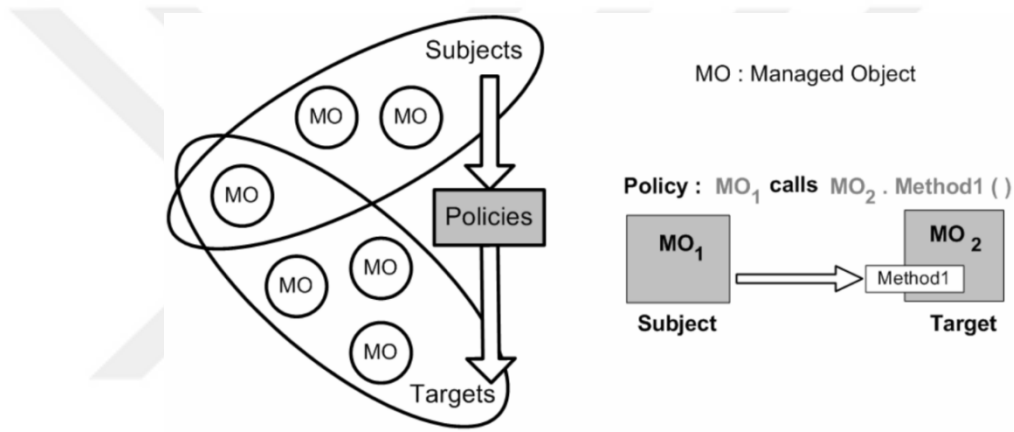
3.2 POLICE PBM Yazılımı

Bu bölüm, ağırlıklı olarak Ponder dili [36] ve diğeri dillerden bazı özellikler içerecek şekilde geliştirdiğimiz PBM çatısını tanıtır. İzleyen bölümlerdeki çalışmaları anlamaya yardımcı olması için bu tanıtıcı bölüm konulmuştur.

adaptasyon katmanıdır. Politikaların tetiklenme koşullarının sağlanıp sağlanmadığını izleyebilmek için PC'ler Olay Servisine (ES) kaydolup sistemdeki nesnelerin durumlarını ve olayları dinlerler. PC'ler, ES'nin uyarısıyla koşul bölümleri sağlanan politikalardaki eylemleri, kendi sorumluluğundaki aktör nesnelere yaptırırlar.

3.2.2 POLICE politika tanımlama dili

Police dili, literatürde ayrı olarak kullanılan politikaların koşul kısımları ile olay kısımlarını birleştiren özgün bir yaklaşım getirir. Aktörler (subjects), kullanıcılar ve aktif yazılım bileşenlerine karşı düşer. Hedefler (targets) ise aktörlerin erişmek istediği nesneleri simgeler. Hedeflerin metodlarının aktör nesneler tarafından çağırılması, bir erişim eylemini temsil eder (Şekil 3.6).



Şekil 3.6 : Nesneye dayalı Police politika modellemesi.

Politikalar, bu metod çağrılarının yönetilmesi için tanımlanan kurallardır. Police dili, bu nesneye dayalı modele göre, PBM sistemi gereklerini (Bölüm 3.1.2) aşağıdaki gibi karşılamaya çalışır:

- Yetki (authorization) ve Emir (obligation) tipinden politikalar kullanılır.
- Negatif politika olmadığından bazı çakışma türleri önlenir (Bkz. Bölüm 3.2.5).
- Yönetilen sistemdeki olayları izleyip oluştuklarında bazı eylemler gerçekleştirebilmek amacıyla sadece Emir politikalarının değil, Yetki politikalarının koşul ifadelerinde de Olay tipinden kısıtlar kullanılır.
- Olay tipinden kısıt içeren koşul ifadeleri sayesinde, periyodik olarak veya olaylarla tetiklenen yönetim eylemlerine olanak sağlanır.
- Politikalar, yönetilen sistemin detaylarından bağımsız olarak uygulanır.

- Politikaların geçerliliğini sınırlandırmak için kısıt ifadeleri kullanılır. Ayrıca politikalar tanımlanırken, yürürlük tarihleri de tanımlanır. Politikalar, yürürlük tarihine göre devreye alınır veya kaldırılırlar.
- Literatürde çokça kullanılan Domain kavramı kullanılarak yönetilen varlıklar gruplanıp bu gruplara ilişkin politikalar tanımlanabilir.
- Politikaların yeniden kullanımı desteklenir.
- Yönetilen sistem arayüzü (NMI) sayesinde, politika dili yönetilen sistemden bağımsızdır. Politika uygulamak için yönetilen sistemde değişiklik gerekmez.
- Negatif politikalar desteklenmez. Dil tanımında semantik çakışmalar için doğrudan destek yoktur. Bunun yerine semantik çakışmalar, E-Code adını verdiğimiz betik tabanlı bir mekanizma ile kotarılır.

3.2.2.1 Dil söz dizimi

Çizelge 3.1’de açıklanan dil sentaksında koyu kelimeler dilin literalleridir (keyword). Birbirine alternatif ifadeler parantez içinde | simgesi ile ayrılır. Seçimlik bilgiler ise

Çizelge 3.1 : Police politika söz dizimi.

Tip	Söz dizimi
Yetki Politikası	<pre> inst cando <i>politikaAdı</i> { [actor domainScopeExpression;] target domainScopeExpression; action action-list; //target nesne üzerinde izin verilen eylem when constraintExpression; // iznin gerçekleşme şartı } </pre>
Emir Politikası	<pre> inst mustdo <i>politikaAdı</i> { [actor domainScopeExpression;] target domainScopeExpression; action obligationActionList; //target nesne üzerinde yürütülmesi gereken eylem when constraintExpression; // zorunluluğun gerçekleşme şartı } </pre>
Kısıt ifadesi	<pre> ((always never) (timeExpression conditionExpression eventExpression)*) </pre>

köşeli parantez içine alınır. Politikaların yorumlanması, SableCC [85] adlı derleyici derleyicisi ile üretilen derleyici ile yapılır (Police politika derleyicisini üreten BNF söz dizimi EK A’da verilmiştir). Politikalarda aktörler ve hedef nesneler, DSE (domain scope expression) kavramı (Bkz. Bölüm 3.2.3.1) kullanılarak tanımlanır. Hedef varlıklarda metod çağrılarını olarak modellenen eylemler aslında, Emir politikalarında hangi eylemlerin gerçekleştirilmesi gerektiğini, Yetki politikalarında ise hangi

eylemlerin gerçekleştirilmesine izin verildiğini belirtir. Eylem kısmında, bir veya virgül ile ayrılmış bir çok eylem bulunabilir. Kısıt bölümleri, politikaların uygulanabilirliğini, zaman, olaylar ve yönetilen nesnelere göre tanımlar. Gizli aktörler kullanılarak (*impersonal policy*), bazı koşullar oluştuğunda istenen eylemlerin yürütülmesi sağlanabilir. Bu yüzden politikaların aktör alanı seçimliktir. Negatif politikalar [53, 86, 87] yerine, *never* ve *always* kelimeleriyle eylemler aktör bazında sınırlandırılabilir. Her eylem ön tanımlı olarak yasaktır (*Permission-Based Regime*).

Yetki (*cando*) politikaları (Örnek Şekil 3.7) aktörlere hedef nesneler üzerinde çeşitli eylemler için (metod çağrıları) yetki tanımlar.

```
type cando Login (actor A, target T, validTI, int maxLogOff) {
  action T.login();
  when Times.within(validTI) and A.dailyLogoffCount < maxLogOff;
}

inst cando /testPolitikaları/loginPolitika1 =
  Login(/student/u54,LabServer/server1, 0900-1700,10);

/* u54'ün bütün öğrencilerinin server1 sunucusuna 09:00-17:00 arası ve
günde en fazla 10 kez girmeye hakkı vardır.*/
```

Şekil 3.7 : Örnek yetki politikası.

Emir (*mustdo*) politikaları belli koşullar oluştuğunda ve/veya belli olaylar oluştuğunda yürütmesi gereken eylemleri tanımlar. Koşullar oluştuğunda, aktöre gerekli iznin tanımlı olup olmadığına bakılmadan eylem yürütülür (Permission-based regime). Şekil 3.8'de, Ponder'daki muadillerine [36] karşı gelen emir politika örnekleri vardır.

```
// 10 dakikadan uzun bekleyen iş yükü çöpe atılacak
type mustdo ExpiredJob (target t){
  when t.waitTime >= Time.minute(10);
  action t.discard();
}

// Scheduler'ın ortalama cevap süresi 10 saniyeden düşük ise
kuyruktan iş seçim stratejisini First Come First Served yap.
type mustdo changeSchedulingStrategyT(target scheduler) {
  action scheduler.setStrategy(Scheduler.FCFS);
  when scheduler.art()<10;
}

//Güvenlik Politikası
inst mustdo filterIP {
  actor a = /Router/router1/routingDaemon;
  target /Router/router1/packetQueue;
  action discard(IPPacket);
  when packetReceived(IPPacket) and
    (IPPacket.getSource()=='192.168.1.5');
}
```

Şekil 3.8 : Örnek emir politikaları.

3.2.2.2 Kısıt ifadeleri

Politikaların hangi koşullarda geçerli (valid) olduğunu ve yürütüleceğini belirtir. Politika kısıtlarını ifade etmede OCL [88] kullanılır. Kısıt ifadeleri aşağıdaki ifade tipleri kullanılarak oluşturulabilir:

- Aktör/Hedef durumu: yönetilen nesnelerin özelliklerine dayalı kısıt koşulları
- Eylem/Olay parametreleri: politikadaki eylem veya olay parametrelerinin değerlerine dayanan kısıt ifadeleri
- Zaman
- Olaylar: basit veya karmaşık olay ifadeleri (Bkz. Bölüm 3.2.3.3 ve 3.2.2.3).

Diğer politika dillerinin [63, 89] aksine tüm politikalar olaylarla tetiklenebilir.

3.2.2.3 Olay ifadeleri

Sistem olaylarını ifade edebilmek için, PDL (Policy Definition Language) [59] dilinde kullanılan yapılar (olayların sıralamasını, birlikte oluşmasını, n kere oluşmasını ifade eden $e_1 \rightarrow e_2$, $e_1 | e_2$, $n * e$, $e_1 \&\& e_2$ gibi yapılar) kullanılır. Olaylar, gerçek ve sözde olay olmak üzere iki çeşittir. Gerçek olaylar, yönetilen sistemde oluşan gerçek olayları, Sözde olaylar ise yönetilen sistem bileşenlerindeki durum değişikliklerine karşı gelen türetilmiş olaylardır. Örneğin, “İş Sıralayıcı kuyruğundaki iş sayısı 100’ü geçince”, “RMS1.bilgisayar4.işYüküOranı>%50 olursa” vb. koşulları izleyebilmek için tanımlanırlar. Diğer dillerde bu işlem, ayrı bir politika tanımlayarak politika uygulama etmenlerine yaptırılabilir. Police modelinde ise yönetilen nesneler arası etkileşimler, ayrıca bir politika tanımlamadan da sözde olay kavramı kullanılarak proaktif izlenebilir. Bu amaçla Bölüm 3.2.3.3’de açıklanan *notifyBefore* ve *notifyAfter* fonksiyonları bulunur. *Sözde olaylar*, sistem olayları ile PDL operatörleri yardımıyla birleştirilerek karmaşık olay ifadeleri oluşturulabilir. Ayrıca modelimizde, Ponder dilinin olay ifadelerinin ayrı tanımlanıp politikalarda tekrar kullanılmasını (reuse) sağlayan özelliği de aynen kullanılmıştır. Örnek:

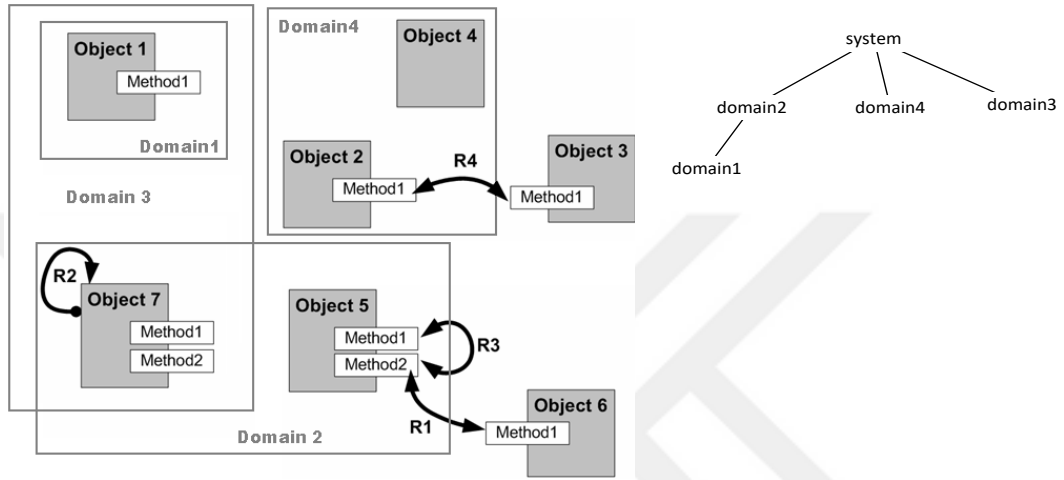
```
event resourceStarvation(rms,type)=(queuesFull(rms)->resListEmptied(type));  
event resendJob = Timer.at("28:02:2015", "23:59:00");
```

3.2.3 POLICE politika uygulama bileşenleri

Bu alt bölümde, Police çatısında politikaları yürütmek için kullanılan modüller, hiyerarşik olarak yukarıdan aşağıya doğru tanıtılmıştır.

3.2.3.1 Etki alanı servisi

Etki Alanı Servisi (DS: Domain Service), politikalar, yöneten/yönetilen varlıklar ve yönetilen varlıklar arası semantik ilişkilerin bilgi modelini tutar. Her semantik ilişkiye ait, E-Code adı verilen ve çakışma çözmede kullanılan bir betik tanımlanır. Bilgi modelinde, yönetilen nesneleri hiyerarşik olarak gruplamak için “domain” kavramı, [90]’de tanımlandığı gibi kullanılır. Örnek bir domain ağacı Şekil 3.9’da gösterilmiştir.



Şekil 3.9 : Örnek SIM senaryosu ve semantik ilişkiler.

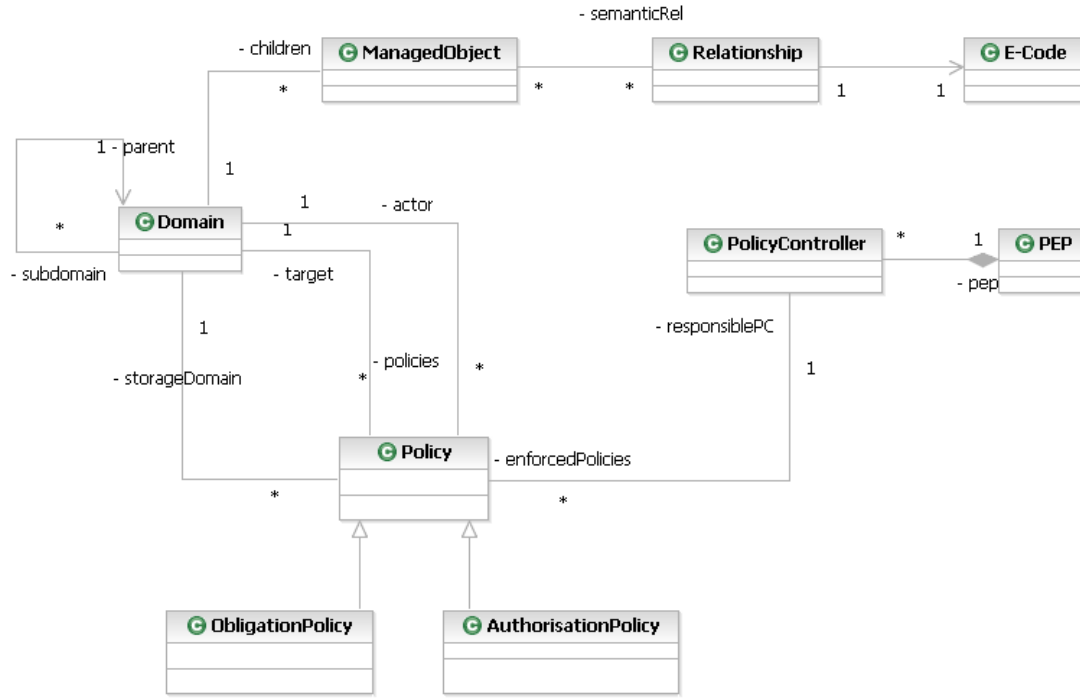
Etki alanları hiyerarşik isimlendirilir (Örnek; /domain1/domain2) [91]. Politika tanımlanmadan önce, politika ile ilişkili bilgi modeli tanımlanmış olmalıdır. Bilgi modelinde yönetilen nesneleri tanımlamak için (managed objects), *Model Sınıfı* adlı (Şekil 3.10) ve yönetilen nesnenin özelliklerini (attributes) ve metodlarını tanımlayan bileşenleri kullanır. Metodlar, yönetilen nesne üzerinde yürütülebilecek olan eylemleri simgeler.

```
Class JobQueue {  
    //özellikler  
    int capacity;  
    //metodlar  
    public addJob (Job j);  
    public popJob (int jobId) throw Exception;  
}  
//statik semantik ilişkiler  
Relation Relation1 between  
JobQueue.addJob and Trader.evaluateRequest with Rel1_E_Code;
```

Şekil 3.10 : Bir RMS iş kuyruğu için model sınıfı ve semantik ilişki örneği.

Model Sınıflarımız, DMTF’nin CIM şemalarına [92] uygundur. SIM bilgi modeli, PEP’lerin yönetilen nesneleri keşfetme mekanizması ile veya operatör müdahalesi ile

güncel tutulur. Yönetilen sistemlerin kendisi, yönetilecek nesneleri, bunlar arasındaki semantik ilişkileri ve bu semantik ilişkiye ait E-Code'ları içeren bir bilgi kümesini (profil), ilgili PEP'e verirler. Alınan bu profil, SIM'e eklenir. Ayrıca, sistem yöneticisi, bunlara ek olarak elle başka semantik ilişkiler ve E-Code'lar da tanımlayabilir. Sistemin bilgi modeli (SIM: *System's Information Model*) adını verdiğimiz modelin daha geniş hali ise Şekil 3.11'de verilmiştir ve politika uygulamanın diğer bileşenlerini de içerir.



Şekil 3.11 : Policy'in geniş bilgi modeli.

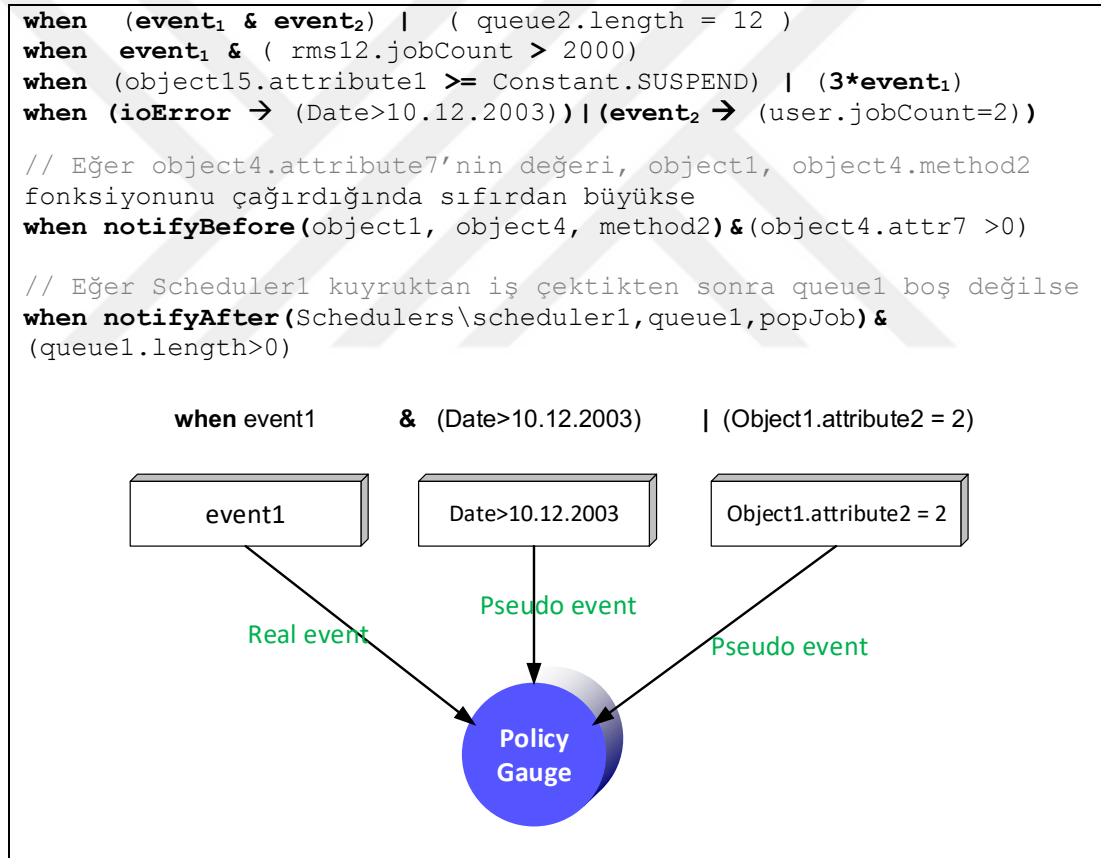
Bu modelden de görüleceği üzere politikalar, domain'ler altındaki yönetilen nesneler hakkında tanımlanır.

3.2.3.2 Politika uygulama noktası (PPEP)

PPEP, politika dağıtımı ve izleme amacıyla Politika Suncusu ile (deployment bus), politikaları uygularken diğer PPEP'lerle (event bus), Yönetim Arayüzü üzerinden ise yönetilen sistemin kendisi ile haberleşir. PPEP'in PC (Policy Controller) bileşenleri, politikaları uygular. PC, kendi sorumluluğundaki aktörlere ilişkin tüm politikaları uygular. PPEP içindeki RC (Resource Controller), aktörlerin hedef nesnelere erişim girişimlerini yakalayıp ilgili PC'lerden, gerekli iznin varlığını sorgular.

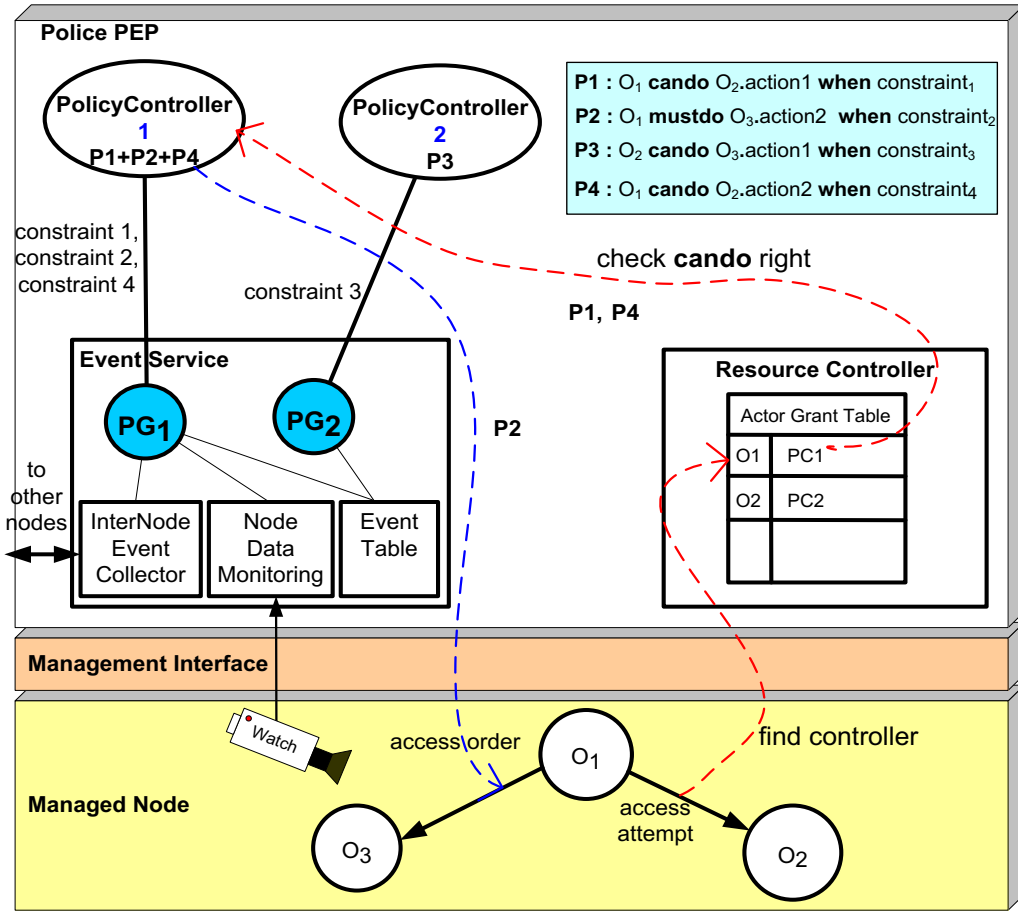
3.2.3.3 Olay servisi (ES)

Yönetilen platformdan olayları toplayıp ilgilenen PC'lere ve diğer PPEP'lerdeki ES'lere gönderir. Diğer ES'ler birlikte, dağıtık bir gözleme sistemi oluştururlar. Bütün Politika tiplerinin kısıt bölümlerinin sağlanıp sağlanmadığının testi için, bu kısıt bölümleri, ES olay ifadelerine dönüştürülür (Bkz. Bölüm 3.2.2.3). Olaylar oluştuğunda yetki politikalarını tetiklemek mümkün olur. Bir PC, kontrolündeki politikaların kısıt bölümlerinin izlenmesini sağlamak için ES'ye kayıt yaptırır ve ES içerisinde politikalar için, PG (Policy Gauge) adı verilen etmenler oluşturulur. PG, ilişkili olduğu politikanın kısıt bölümünün sağlanıp sağlanmadığını sürekli izler ve değişiklikleri PC'ye bildirir. Kısıt bölümü örnekleri aşağıda verilmiştir. Ayrıca Şekil 3.12, Police politika kısıt bölümlerinin PG'lere nasıl dönüştüğünü gösterir.



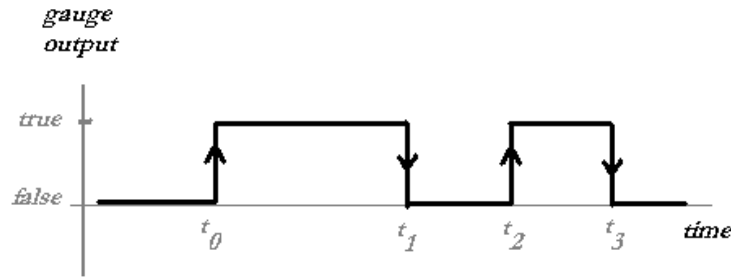
Şekil 3.12 : Police bileşik olay ifadesi örnekleri ve PG olarak gerçekleşmesi.

ES'nin politika uygulama sürecindeki yeri Şekil 3.13'de gösterilmiştir. Politikaların kısıt bölümlerinde, başka bir PEP'in denetimindeki bir nesnenin veya olayın adı belirtilebilir. Bu durumda, yerel ES, uzak PPEP'in olay servisine (olay yolu üzerinden) kaydolarak ilgilendiği değişimleri dinleyebilir.



Şekil 3.13 : Politika uygulamada, PC, RC ve ES arasındaki etkileşim.

Kayıt ile oluşan PG, olay oluştuğunda (kısıt koşulunun sağlandığını) PC'yi uyarır. Şekil 3.14'de bir PG çıktısının zamanla değişimi gösterilmiştir.



Şekil 3.14 : Örnek PG çıktısı.

Şekile göre, $[t_0-t_1]$ ve $[t_2-t_3]$ zaman aralıklarında kısıt ifadesi sağlanır. PG çıktısı *true* olduğunda, ilişkili politika bir yetki politikası (cando) ise eğer politikanın aktörü girişimde bulunursa politikada belirtilen eylemi gerçekleştirilebilecektir. Eğer politika emir (mustdo) tipinden ise PC, PG'nin *false* $\rightarrow$ *true* geçişlerinde, aktörleri, politikada belirtilen eylemi gerçekleştirmeye zorlar, *true* $\rightarrow$ *false* geçişlerinde ise eylem devam ediyorsa durdurmaya zorlar.

Olay Servisi içindeki gerçekleştirme yapısı, Ponder modelindeki gibi kurulur. *Farklılık olarak*, gerçek olayların yanında sözde olaylar da izlenebilir ve diğer PPEP’lerdeki uzak olaylar da dinlenebilir. Bir PC, ES’ye kayıt olduğunda, kısıt ifadesine karşı gelen bir *olay ağacı* oluşturulur. Ağacın düğümleri, ($e1 \rightarrow e2$) or p_e3 kısıt ifadesinde yer alan parçaların, gerçek ve sözde olaylara dönüştürülmesi sonucunda oluşturulur (Şekil 3.12’de bunun örnekleri verilmişti). Olay ağacının düğümleri, politikanın kısıt ifadesinin parçalarına karşı düşer. Ağacın düğümlerinden, gerçek olaylara karşı düşenler, ES’deki olay tablosuna kayıt olup olay oluştuğunda uyarılmayı beklerler.

```

graph TD
    subgraph Event_Service [Event Service]
        direction TB
        or((or))
        arrow((→))
        PG((PG))
        p_e3[p_e3]
        ET[Event Table]
        
        ET -- "eventOccured (event, params)" --> MI[Management Interface]
        MI -- "getValue (MO, property), getResult (MO, method, params)" --> MO((MO))
        
        or -- "notify" --> PG
        or -- "notify" --> arrow
        arrow -- "notify" --> ET
        arrow -- "notify" --> p_e3
        p_e3 -- "getValue (MO, property), getResult (MO, method, params)" --> MO
    end
    
    InterNodeBus[Inter-node Event Bus] -- "wait" --> ET
    PG -- "notify" --> PC((PC))

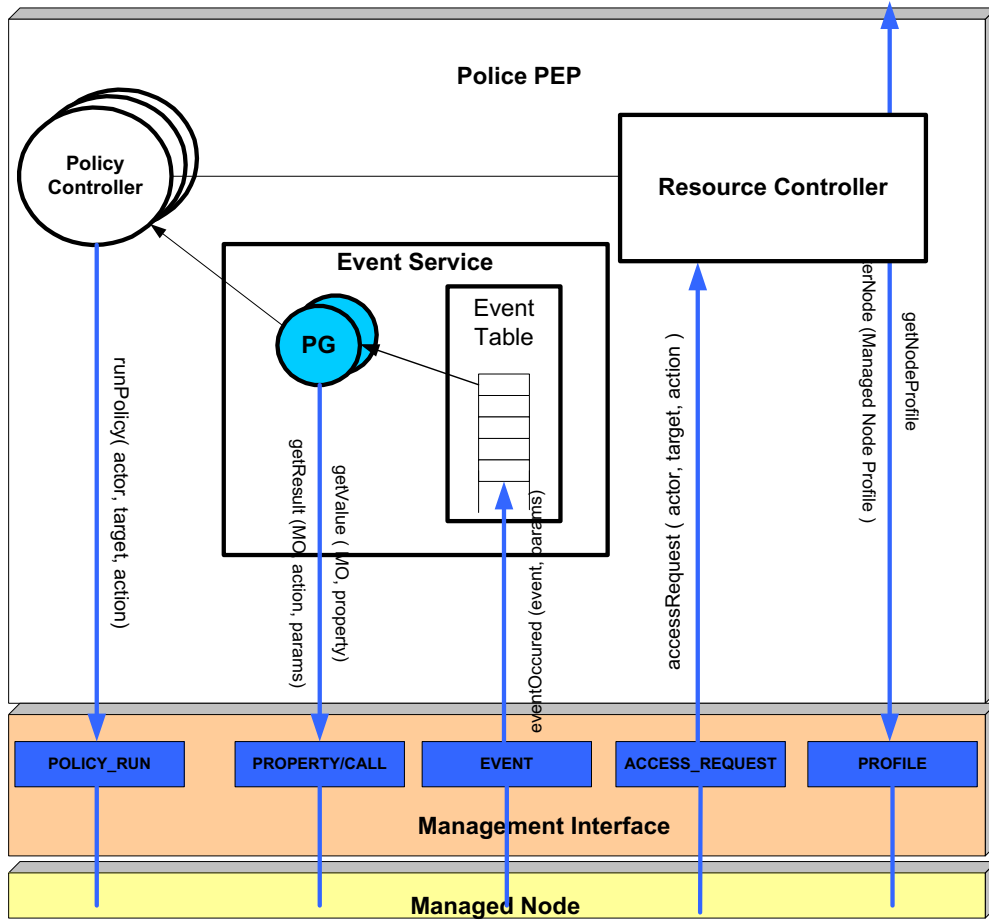
```

51

oluşturmadan önce e_l oluşmuş ise bir olay üretir. Eğer e_l başka PEP altındaki uzak bir olay ise oluşup oluşmadığı, olay yolu (event bus) üzerinden başka bir PEP'in olay servisi üzerinden dinlenir.

3.2.3.4 Yönetim arayüzü (MI)

MI (Management Interface), Police politika uygulama elemanları ile yönetilen sistem arasında sabit fonksiyonlara sahip bir uyarılma katmanı görevi görür (Şekil 3.16).



Şekil 3.16 : Yönetim Arayüzü üzerindeki hizmet erişim noktaları (SAPs).

Arayüzde tanımlı yetenekler şunlardır:

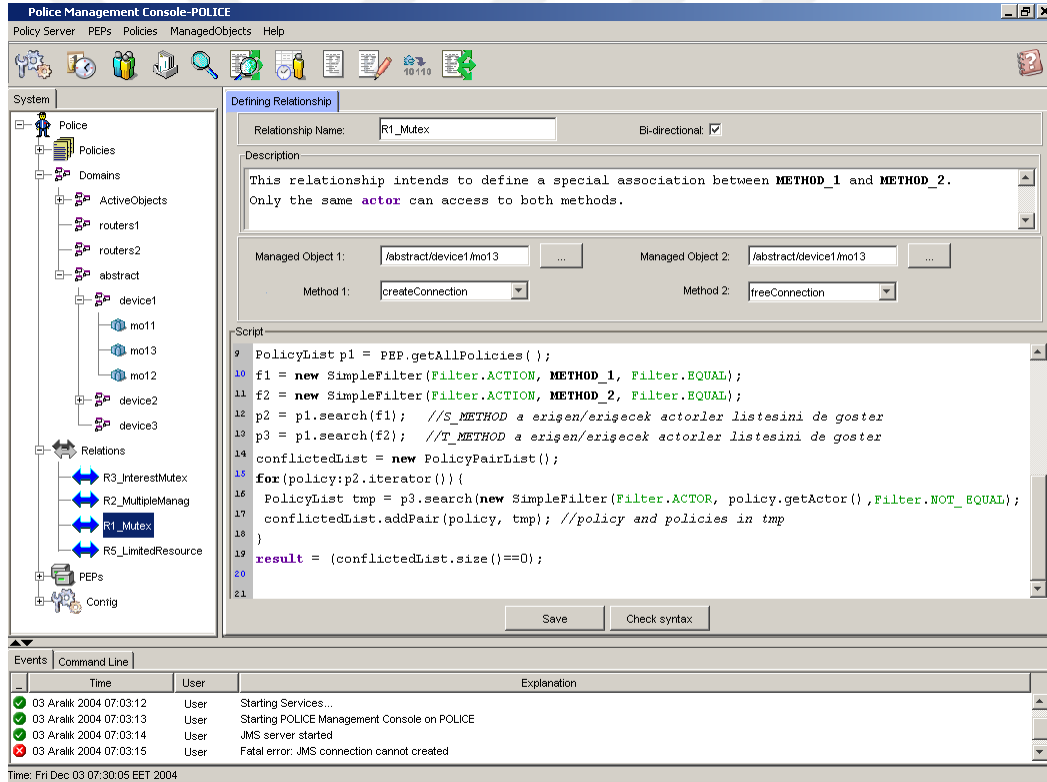
- **checkAccessRequest:** PEP'e aktörün, erişim izin olup olmadığı sorulur,
- **setGetParameter:** PEP, yönetilen nesnenin bir özelliğine değer ataması, özelliğin değerinin öğrenmesi,
- **policy_Run:** PEP, bir yönetilen nesne fonksiyonunu çağırır,
- **profile:** PEP'e yönetilen nesneleri ve semantik kısıtları kayıt ettirir,
- **event:** yönetilen düğümdeki herhangi bir olayı PEP'e bildirmek .

Yönetilen platformdaki nesneler üzerinde yürütölmek istenen eylemler, MI tarafından yönetilen platforma özöü çağrılara dönüştürölür.

3.2.4 POLICE çatısının geröeklemesi

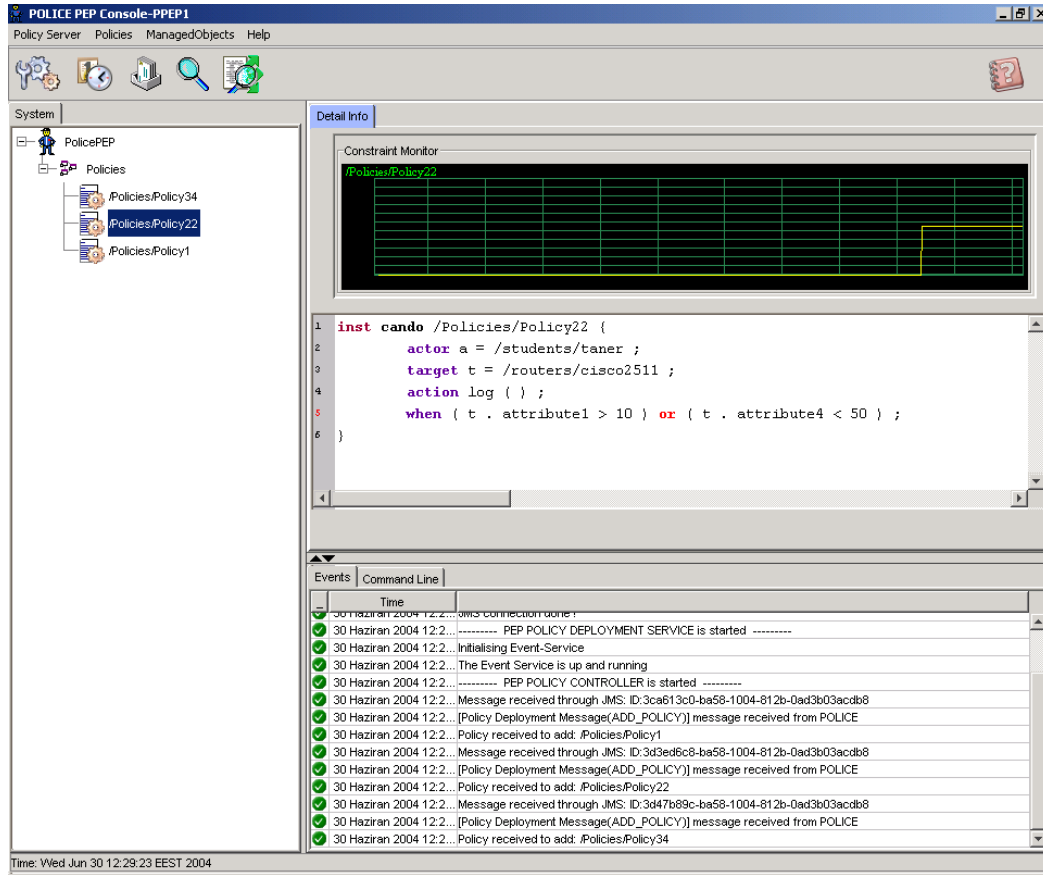
Politika nesneleri, JNDI [93] ile erişilen, Apache Directory Server [94] LDAP dizininde saklanır. Domain servisi nesnelerini oluşturabilmek için, LDAPv3 dizin servisi [95] kullanılarak SIM classlarından oluşun LDAP şema tanımlanmıştır. Global Olay Servisinin geröeklemesinde OpenJMS [96, 97] kullanılmıştır.

Politikalar, politika editorö kullanılarak tanımlanır, derlenir ve domain servisinde saklanır. Daha sonra politikaları dağıtmak ve yönetmek için yönetim konsolu kullanılır. Yönetim konsolu ile Police çatısı altındaki PC'ler, Olay Servisleri, vb. bileşenler de yönetilebilir. Şekil 3.17'de ana konsola ait göröüntöler verilmiştir.



Şekil 3.17 : Politika Yönetim konsolu ekran göröüntöleri.

Şekil 3.18 ise üç politikayı uygulayan bir PEP'e ait konsol penceresini göstermektedir. Politikaları derleyecek derleyici, SableCC [85] isimli nesneye-dayalı derleyici geliştirme çatısı ile geliştirilmiştir. Police çatısı, politikaları nesneler olarak geröekler (policy as objects). Politika derleyicisi kullanılarak, tanımlanmış her politikaya karşı



Şekil 3.18 : POLICE PEP izleme konsolu.

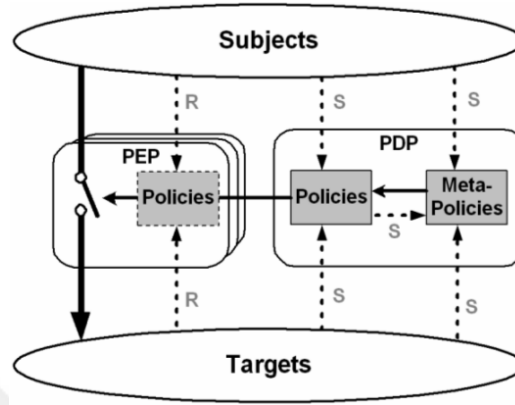
bir Java sınıfı üretilip kaydedilir. Politikayı uygulamak için, bu sınıftan bir politika nesnesi (PO) türetilip PEP'lere gönderilir. PPEP'in yönetilen sistemle arayüzünü (MI) gerçekleymesinde (Bölüm 3.2.3.4), WBEM Services [98] yaklaşımı kullanılmıştır.

3.2.5 POLICE çakışma kotarma yöntemi

Bu bölümde, geliştirdiğimiz Police çakışma kotarma yaklaşımı [4, 8], Ponder modeli ile karşılaştırmalı olarak açıklanmaktadır. Politikalar birbirlerini etkileyerek yönetilen sistemi istenmeyen durumlara sürüklüyorlarsa çakışmalar oluşabilir [53]. Çakışmalar, temel olarak iki gruba ayrılır:

Modalite Çakışmaları [53, 86]: Yönetilen aynı bir nesne grubuna uygulanan hem pozitif hem de negatif yetki ve yükümlülük politikaları olduğunda ortaya çıkar. Politikaların söz dizimsel analizi ile politikalar uygulamaya konulmadan önce tesbit edilebilirler. Tesbit edildikten sonra, politikalara atanmış öncelik (precedence) değerleri kullanılarak çakışmalar otomatik olarak çözülür.

Semantik Çakışmalar (Uygulama uzayına özgü çakışmalar): Semantik çakışmalar, politikanın, sistemdeki sınırlı kaynaklar, sistem topolojisi veya sistemin diğer politikaları gibi bilgilerle olan uyumsuzluklarıdır. Çakışmanın bir tanımı verilmeden, sadece politikalar incelenerek tesbit edilemezler. Literatürde *Meta-politika* kavramı [53, 86], izin verilen politikalar için kısıtlama tanımlamaya yarayan bir metod olarak önerilmiştir (politika hakkında politika). Şekil 3.19’da, meta politikaların



Şekil 3.19 : Meta-politika tabanlı çakışma kotarmanın yönetimdeki yeri.

politika uygulama aşamaları içindeki değerlendirme yeri gösterilmiştir. Şekilde kesik çizgiler, ilgili modülün girdilerini, koyu çizgiler ise kontrol akışlarını göstermektedir (R: yönetilen nesnelerin durumları ile ilgili yürütme zamanı verisi. S: politika tanımlarına ilişkin statik veri).

Meta-politikalar merkezde, sadece statik veriler (politikaların kendileri) kullanılarak incelenir. İncelemenin sonucuna göre bazı politikalar semantik çakışma ihtimali dolayısıyla elenirler. Ancak, meta-politikalar, problemi her zaman çözemeyebilirler. Yönetici, meta-politikaları bütün olası yürütme zamanı durumları için tanımlamalıdır ve bu da başka çakışmalara sebep olabilir. Yöneticinin, yönetilen sistemdeki çakışmaya sebep olabilecek bütün dolaylı ilişkileri düşünebilmesi her zaman mümkün olamayabilir. Ayrıca, çoğu durumda politikaların birbiri ile ilişkili olsa bile yürütme aşamasında çakışmaların şartları oluşmayabilir. Bu nedenle, statik verilerin incelenmesine dayalı meta-politikalar kullanılarak tesbit edilmiş olan çakışmalar, yürütme esnasında oluşmayabilir.

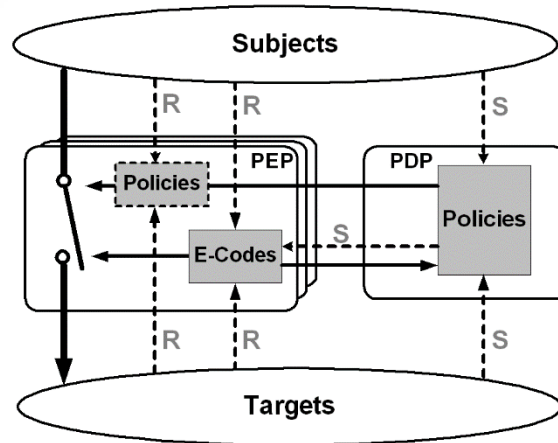
Police, hem politika uygulama mimarisinin teknik detaylarından hem de politika dilinin formatından bağımsız olan bir semantik politika-çakışması kotarma modeli [86] içerir. Meta-politika tabanlı yaklaşıma bir alternatif olarak, yönetilen sistem ile nesneler arasındaki ilişkilere dayalı bir politika çakışma yönetim mekanizması

geliştirilmiştir. Police çakışma yönetim modeli, SIM’de tanımlı olan, yönetilen nesneler arasındaki uygulamaya özel (semantik) ilişkileri (R_i) kullanır (Şekil 3.9’da örneklendirilmiş olan).

Çıkış noktası, aktörler ve hedef nesneler arasındaki semantik ilişkiler değerlendirilerek, bu aktör ve hedef nesnelere uygulanacak politikaların aralarındaki, doğrudan veya dolaylı ilişkilerin de belirlenebilmesidir. Bu değerlendirme işlemi E-Code (Evaluation Code) adını verdiğimiz özel kod parçaları ile yapılır. E-Code, yönetilen sistemin kendisi tarafından sağlanır veya operatörler tarafından kodlanır.

Çakışma analizi mantığı, yönetilen sistemin kendisinin içine gömülmüştür. Operatörler politikaları tanımlarken, olası semantik çakışmaları dert etmek zorunda değildir. E-Code’lar, PEP’ler içinde çalıştırılır ve çakışma analizi merkezi değil, PEP’ler üzerinde, dağıtık yapılmış olur.

Şekil 3.20’de gösterildiği gibi, PEP’lerde çalışan E-Code’lar, çakışma analizi için hem statik hem de yürütme zamanı verilerini kullanır. E-Code’lar, politika tanımları gibi statik verilere ek olarak, yönetilen nesnelerin öznelilikleri, yönetilen sisteme ait olan yürütme zamanı verilerini de kullanırlar.



Şekil 3.20 : Police çakışma kotarma modeli.

PDP üzerinde merkezi ve statik çakışma analizi yapan ve meta-politika kullanan Ponder’ın aksine, Police, çakışma kararının verilmesi yürütme anına kadar erteler. Ayrıca, E-Code’lar meta-politikalara göre daha esnek ve daha genişleyebilir yapıdadır. Çünkü çakışma kotarma modeli politika dilinden bağımsızdır. E-Code’lar, Java, Tcl gibi dillerde yazılabilir. Police gerçekleştirilmesinde Java BeanShell [99] tercih edilmiştir. E-Code’u daha iyi anlatabilmek için meta-politika kavramı ile örnekler üzerinden karşılaştırmak faydalı olacaktır. Aşağıdaki örnek üzerinden karşılaştırma verilmiştir:

Örnek: (Separation of Duty): “Farklı aktörlerce yapılması gereken iki işi, aynı aktöre yaptıran politika çifti olmamalıdır.” [86], [100].

Bu tip çakışmayı yakalayan Ponder meta-politikası (Şekil 3.21), aynı kişinin hem *approve* hem de *submit* metodlarını çağırmasını yasaklar. *Budget* adlı hedef nesnenin *submit* ve *approve* eylemlerini bulunduran politikalardan aynı aktörlü olanlar bulunursa *conflictInBudget* olayı tetiklenir.

```
inst meta budgetDutyConflict raises conflictInBudget(z) {  
  [z] = this.policies -> select (pa, pb |  
    pa.subject -> intersection (pb.subject)->notEmpty and  
    pa.action -> exists (act | act.name = “submit”) and  
    pb.action -> exists (act | act.name = “approve”) and  
    pb.target -> intersection (pa.target)->oclIsKindOf (budget))  
  z -> notEmpty ;  
}
```

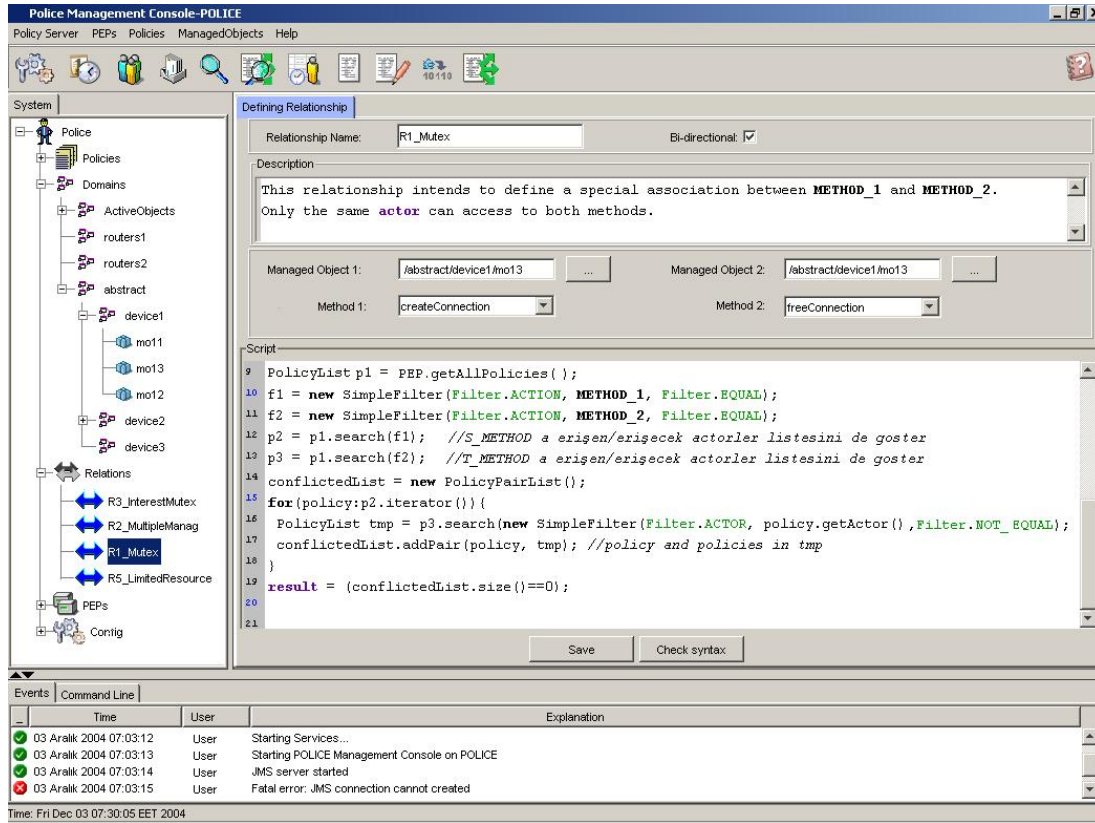
Şekil 3.21 : Ponder meta-politikası (görev ayrımı semantiği için)

Aynı semantik çakışma için Police modelinde tanımlanan ve PEP tarafından yürütülecek olan E-Code ise Şekil 3.22’de verilmiştir.

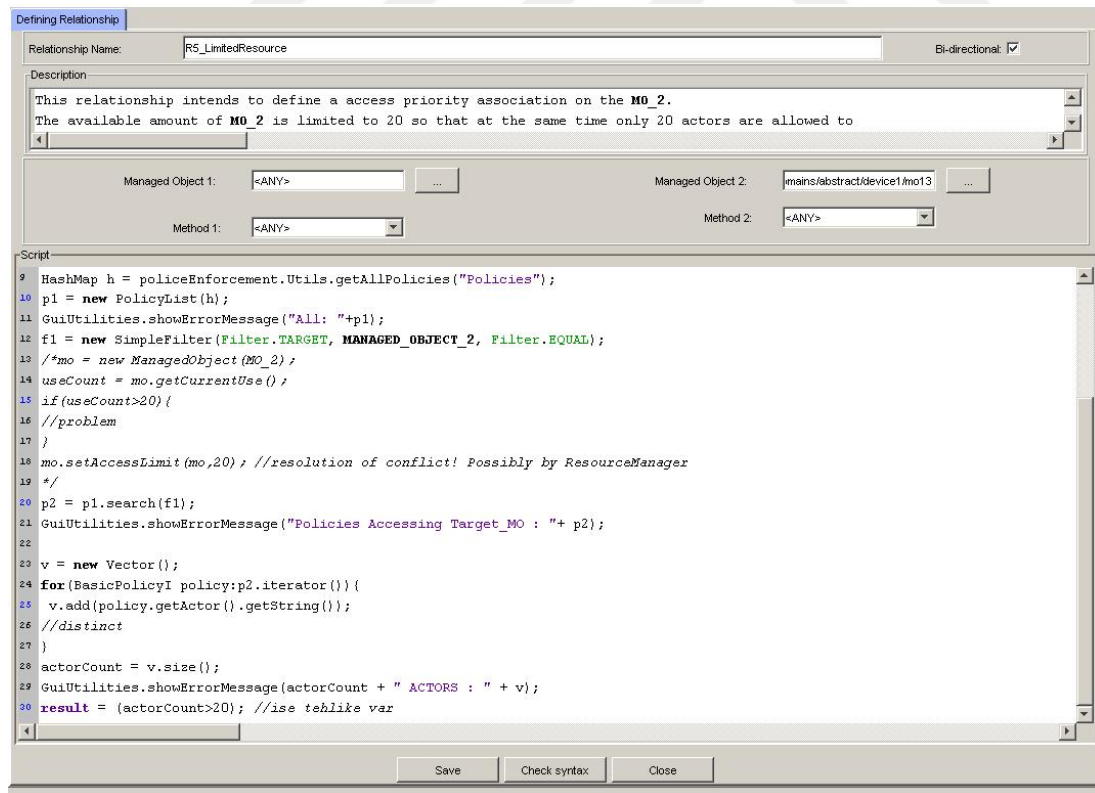
```
boolean EC_DutySeperation( Method submit, Method approve){  
  PolicySet p1 = PDP.getAllPolicies();  
  f1 = new Filter(ACTION, EQUAL, submit);  
  f2 = new Filter(ACTION, EQUAL, approve);  
  p2 = p1.search(f1); // eylemi submit olan politikaları bul  
  p3 = p1.search(f2); // eylemi approve olan politikaları bul  
  conflictedList = new PolicyPairList();  
  // p2 ve p3’ten aynı aktöre sahip olmayan politikaları bul  
  for (politika:p2.iterator()){  
  
    PolicyList tmp = p3.search(new Filter(SUBJECT, NOT_EQUAL,  
    politika.getSubject() ));  
    //match politika with tmp politikalar  
    conflictedList.addPair(politika,tmp);  
  }// for  
  return (conflictedList.size()==0); // çakışma tesbit sonucunu döndür  
}
```

Şekil 3.22 : Duty separation tipinden semantik çakışma için E-Code.

Geliştirilen prototipte, semantik ilişkilere ait E-Code yazmak için PMC konsolunda bir E-Code editörü sağlanır. Söz dizimi doğrulanan E-Code’lar ilgili PEP’e gönderilir. E-Code editörü arka planda BeanShell adlı yorumlayıcıyı barındırır. Aşağıdaki şekiller, çeşitli semantik çakışmalar için E-Code örneklerini göstermektedir.



Şekil 3.23 : Single manager tipinden semantik çakışma için E-Code.



Şekil 3.24 : Sınırlı kaynak semantik çakışması için E-Code tanımlanması.

Diğer örnekler [86] makalemizde bulunabilir.

4. HESAPLAMA SİSTEMLERİNİN PBM İLE YÖNETİLMESİ

Bu tez çalışmasının amacı, YBH sistemlerinin yönetim ihtiyaçlarını araştırmak ve PBM'in bu konuda nasıl katkılar sağlayabileceğini göstermektir. Bu bölümde, YBH sistemlerinin yönetim ihtiyaçları belirlenmiş; literatürde YBH sistemlerinin PBM tabanlı yönetimi konusunda yapılan çalışmalar incelenerek hangi noktalarda geliştirme ihtiyacı olduğu belirlenmeye çalışılmıştır.

4.1 YBH Yönetimi İçin PBM Kullanılması

Bir YBH sisteminin idame ettirilmesi ve yönetilmesi oldukça zahmetli bir iştir. Kullanılmakta olan çeşitli YBH sistemlerinin gerek iş sıralama mekanizmaları gerekse bileşenlerinin bütünleşik yönetiminde, özellikle elle konfigürasyona dayalı (dinamik değişiklikleri dikkate almayan) bir işletme şekli nedeniyle çeşitli problemler oluşabilir:

- Genelde DQS (Distributed Queuing System)'lar ya manuel olarak uzmanlarca konfigüre edilir ya da özel yönetim araçları kullanılarak yönetilir. Operatorlerin sistem hakkında detaylı bilgisi olmalıdır. Günümüzdeki geleneksel iş sıralayıcıların (Scheduler) yeteneği ve esnekliği arttıkça optimal bir şekilde konfigüre edilmesi gereken parametrelerinin sayısı da artar. Kullanıcı sayısı da artar ve sonucunda yönetim parametresi sayısı da artar.
- Esnek değildirler ve iki önemli noktayı ihmal ederler: Çoğu kullanıcı kaynak ihtiyaç miktarını doğru belirtemez. Kısa işler için verilen tahminler genelde büyük hata payı içerir. Kullanıcı tahminleri yanlış ise işler daha erken biter, kaynakların atıl kaldığı zamam dilimleri oluşur ve kuyruktaki iş yüklerinden boşlukları doldurma (backfill) gerekliliği ortaya çıkar. Özetle iş sıralayıcılar, genelde yanlış bilgilerle çalışmaktadır.
- Tipik kullanıcı, her zaman boştaki maksimum kaynağı ister veya her iş yükü (job) için aynı miktarda kaynak tahsisini ister. Bu ise iyi iş planlamaları doğurmaz.
- Statik bilgiler kullanılarak iş sıralaması yapılırsa sorunlar çıkar. Zira kaynakların gerçek durumu ancak yürütme anında belli olur ve dinamik olarak da değişir.

Eğer iş yükünün hesaplama ve haberleşme karakteristiği önceden biliniyorsa kaynak atamak basit bir eşleştirme işidir. Ama heterojen tipli ve farklı sahipleri olan, dinamik olarak değişen kaynakların olduğu ortamlarda, iş yüklerinin geliş hızı ve gereksinimleri de önceden tahmin edilemez ve geleneksel eşleştirme yöntemleri ile kaynak tahsisi çalışmaz. Bu tip bir eşleştirme problemi için optimal çözüm, NP-complete olarak bilinir. Bu yüzden, tüm çözüm kümesini araştırmak yerine bir *suboptimal* çözüm bulunur [24].

- Mevcut kaynak seçimi stratejileri, kaynakların sadece iş yükü bilgisini hesaba katar ve kaynağın arıza karakteristiğini ihmal eder. Kaynak çökünce üzerindeki tüm işler iptal edilir. İş yüklerinin yeniden başlatılması gerekir. Hem kaynak hem de çalıştırılacak iş seçiminde işlerin doğasını da dikkate almak gerekir.

PBM, hem dinamik değişen koşulları dikkate alabildiği hem de politikalara göre sistem davranışını çok hızlı değiştirebildiği için, YBH sistemlerinin yukarıdaki problemlerini çözme potansiyeline sahiptir. PBM'nin, YBH sistemlerinin yönetim ihtiyaçlarına vereceği cevaplar aşağıda listelenmiştir:

- Yönetim yükünü azaltmalı ve yönetim basitleştirmelidir.

Çözüm: PBM kullanıldığında, politikalar sistem detayını operatörlerden gizler. Operatör, sistemde beklenen davranış tanımlar ve YBH sisteminin yazılımının veya dağıtık uygulamaların çalıştığı sistemlerin detaylarını bilmez.

- Sistem varlıklarının (kaynaklar, kullanıcıların, kuyruklar, vb.) sayılarındaki ve durumlarındaki dinamik değişikliklere adapte olabilmeli, proaktif yönetim desteği sağlamalı.

Çözüm: Bazı sistemlerde sıklıkla kullanılan statik parametre ayarlanması (bazı eşik değerleri, strateji parametreleri, vb.), yönetilen sistemdeki dinamik değişikliklere adapte olamaz. PBM bu adaptasyonu kolaylaştırır. Mesela [101] makalesinde, MasterNode değişimi için eşik değerler, sabit olarak elle tanımlanmaktadır. Bu ise dinamik Grid'lerde verimsiz yönetim yüküne sebep olur. Düğüm sayısı arttıkça bu eşik değerini de artırmak gerekir. PBM bunu sağlayabilir. Politikalar, hedeflenen yönetim amaçlarını ifade ettikleri için proaktif yönetim sağlar ve dinamik bilgileri de kullanabilir. Pek çok kaynak ve koşul için genişleyebilir bir politika dili kullanılarak genişletilebilir yönetim altyapısı elde edilebilir.

- Farklı yönetim uzaylarının (administrative domains) ve araçlarının bulunduğu senaryoları desteklemelidir.

Çözüm: Farklı kişi ve kurumlara ait kaynaklar farklı PBM sistemlerinin yönetim çatısı altında bulunabilir. Her uzayın kendi yönetim politikaları tanımlanabilir. Yönetim uzayları arası kaynak ve iş yükü paylaşım durumları için iki PBM sistemi arasında politika ve SLA pazarlığı mekanizmaları vardır.

- Kaynak sahibinin kaynak, kullanıcının iş yükü, YBH sistemi sahibinin ise sistemin kendisi hakkında kısıt tanımlamasına olanak sağlamalıdır.

Çözüm: PBM ile hem iş yürütme kısıtları hem de sistem kullanım politikaları tanımlanabilir. Lokal ve global politikalar desteklenebilir.

- Sistemlerin heterojen olması ve global durum bilgisini güncel olarak bilmenin zorluğu, kaynak tahsisinin de hiyerarşik olmasını gerektirir. Bu tür hiyerarşik mimariler de yönetilebilmelidir. Farklı seviyede (global/lokal) tanımlanan kural ve kısıtlar arasında uyum sağlamalı, çakışma durumları koterılmalıdır,

Çözüm: Farklı politika konsolları üzerinden aynı veya ayrı PDP'lere tanımlanabilecek olan lokal ve global politikalara destek vardır. Tek PDP olması durumunda politika çakışma çözme yöntemleri, farklı PDP olması durumunda ise politika ve SLA pazarlığı mekanizmaları kullanılabilir.

- Erişim denetimi sağlanmalıdır.

Çözüm: Erişim denetimi, PBM sistemlerinin en çok kullanıldığı uygulama alanıdır.

- Sadece iş sıralayıcı stratejisi değil, iş, kaynak, kuyruk, kullanıcı, vb. tüm varlıklar yönetilebilmelidir. Örneğin, işlerin tamamlanma süresi, kuyruktaki iş sayısı, kaynakların yanıt süreleri, kaynak iş yükü durumu gibi bilgiler, kurallar içerisinde parametre olarak kullanılabilirdir.

Çözüm: PBM, her türlü varlığın yönetilebilir nesne olmasına olanak sağlar.

- YBH sistemlerinin yönetimi, iş yükü planlamasını, kaynakların ve iletişim altyapısının yönetimi ile bütünleşik olarak yapılmalıdır.

Çözüm: Yönetim politikaları, hem yönetilen kaynaklar hem de YBH sisteminin kendisinin davranışları için tanımlanabilir.

- Yönetim sistemi, YBH sisteminin ve iş yüklerinin yazılımında minimum değişikliğe sebep olmalıdır.

Çözüm: Politikalar ile sistemleri yönetmek, yönetilen sistemlerin kodunda herhangi bir değişiklik gerektirmez.

- Gridlerde yaygın olarak kullanılan GT2 gibi araçlar ve iş gönderme mekanizmaları ile kolayca entegrasyon sağlamalıdır.

Çözüm: Örneğin Grid servislerinde kullanılan sertifika tabanlı erişim denetimi mekanizmasında, kullanıcının izinli olup olmadığı politikalarla kontrol edilir. Eğer izinli ise Grid servisi, bir sertifika içine, eylemi yürütmek için gerekli izinleri yerleştirir ve süreleri yazıp imzalayıp kullanıcıya geri verir. PBM, bu mekanizma ile erişim denetimi amacıyla kolayca entegre olabilir.

- Yönetim sürecinin kendisini yönetmeye (self-managing) destek sağlamalı.

Çözüm: Yönetim sisteminin kendisi ile ilgili de politikalar tanımlanabilir. Bu politikalar sayesinde, yönetim bileşenleri de yönetilebilir.

PBM yapısı kullanılarak hem iş yürütme kısıtları hem de sistem kullanım politikaları gibi pek çok kuralı aynı anda dikkate almak mümkün olabilir. Sadece İş Sıralayıcıyı yönetmek yerine YBH sisteminin tüm kaynaklarını birlikte yönetmek mümkün olabilir. Örneğin mevcut sistemlerde, “Lokal işlere öncelik verilsin” şeklindeki basit, statik ve sistemin kendisini ihmal eden konfigürasyonlar kullanılırken, gerçek PBM yapıları ile “Sistem iş bitirme başarımı X değerinden büyükse Lokal işlere öncelik verilsin aksi halde lokal ve uzak işler eşit öncelikli olsun” gibi daha karmaşık ve sistemin çalışma anındaki (runtime) durumunu dikkate alan politikalar tanımlanabilir.

YBH sisteminin tüm parçalarını dikkate alarak yukarıdaki yönetim gereksinimlerini maksimum derecede karşılamaya çalışan ve IETF modeline uygun bir politika tabanlı YBH yönetim sisteminin kullanılabilirliğini göstermek bu tez çalışmasının amacıdır.

4.2 Literatürde YBH Sistemlerinin Politika Tabanlı Yönetimi Çalışmaları

Literatürde, çeşitli YBH sistem çalışmalarında politika kavramından bahsedilmiştir. Ama literatürdeki bu çalışmaların aslında çok azı, gerçek anlamda politika tabanlı yönetime odaklanmıştır. Zira literatürde görülen genel bir karmaşa, YBH sistemindeki

mekanizmaların çalışma modlarına (stratejilerine), yanlış bir yaklaşımla, politika ismi verilmektedir. Bu yanlış kullanıma literatürden birkaç örnek aşağıda verilmiştir:

- “Kuyruktan iş seçme politikası FCFS’dir” şeklindeki yanlış kullanımda aslında, iş planlayıcının FCFS modunda çalışması ayarı belirtilir.
- [102]’de önerilen mekanizmaya *Distributed Fault-Tolerant Scheduling (DFTS) Politikası* adı verilmiştir. Oysa bu bir politika değil, iş sıralarken işlerin birden fazla kopyasını çalıştırarak hata toleransı sağlamaya yönelik mekanizmadır. Aynı durum [103]’de de vardır.
- Benzer olarak, [104] makalesi, “*Close-to-Files*” adı verilen, girdi dosyalarının (işleri yürütecek sitelere) transfer gecikmelerini minimize etmek için önerilen mekanizmaya *CF-Politikası* adını vermiştir. Oysa bahsedilen kavram politika değil, strateji veya çalışma modu ayarıdır.
- PBM kullanmayan [105] makalesi ise gerçek kuyrukların bir kaç çalışma modu arasında geçiş sağlayan basit politikalar içerir. Lokal iş sıralayıcılar için iki öncelik modu vardır. Lokal ve uzaktan gelmiş işler aynı öncelikli ise tek bir kuyruğa, değilse iki ayrı kuyruğa yerleştirilirler (*Local-Job-First* politikası).

Öte yandan literatürde gerçek anlamda PBM kullanmayı amaçlayan örnekler de vardır ancak YBH sisteminin çok küçük bir alanını (genel de iş sıralama kuyruklarını) yönetmeye çalışır. Bazı gridlerde, her bir hesaplama düğümü, sadece kuyruğunda bekleyen iş sayısı ve kullanıcıların tanımladığı süre tahmini bilgilerine dayanarak her bir iş için özel bir planlama yapar. Gerçekte, dikkate alınması gereken daha çok parametre vardır. Her sitenin kendi politikaları ve iş yükleri vardır. Bu nedenlerle, global seviyede, her iş için özel planlama yapmak yeterli değildir. Daha karmaşık bir yöntem gerekir. Örneğin [106]’de önerilen sistemdeki iş planlaması, yürütmelere ait geçmiş kayıtların istatistiksel analizi ve site iş sıralamasının bir simülasyonunu kullanır. Öte yandan sadece iş sıralayıcı kuyruğunu değil, sistemin tamamını (örneğin iletişim mekanizmaları, kullanıcıların sisteme erişimi, kaynaklar, kullanıcılar, vb. parçaların birleşimini) bir bütün olarak ele almak gerekir. Zira sistemin parçaları birbirinden bağımsız değildir. Ama örneğin Grid’lerde kullanılan ağ altyapısının kendi yönetimi, iş yönetiminden bağımsız yapılmaktadır.

İzleyen kısımlar, literatürde YBH yönetimi konusunda politika terimini kullanan çalışmaların incelenmesine ayrılmıştır. Burada incelenen çalışmalardan YBH sistemleri için kullanılabilecek muhtemel politika tipleri, politika uygulamada

kullanılabilecek metrik tipleri ve politika uygulama yapıları da tesbit edilmiştir. Bu bilgiler, ilerleyen bölümlerdeki kendi çalışmalarımızda girdi olarak kullanılmıştır.

4.2.1 Grid ve kümelerin PBM ile yönetimi

[107]'de, politika dilinde tanımlanan grid politikaları, tanımlanan kurallar yardımıyla ağ politikalarına dönüştürülür. Gerçek PBM yönetimi kapsamında, her çeşit politika (grid, ağ, vb.) tanımlayarak YBH sistemini bir bütün olarak yönetmese de QoS tanımlamasına dolaylı da olsa destek sunar.

[108]'de, Semantik Web Servis uygulamaları için politika ve kontrat yönetimi yeteneği önerilmiştir. KPAT (KAoS Policy Administration Tool) yazılımı ile tanımlama, yönetim, analiz, çakışma çözme ve politikaların uygulanması için OWL (Object Web Language) kullanılır. KAoS'u Globus GSI (Grid Security Infrastructure) uyumlu Grid servislerini yönetmede kullanım hedeflenmiştir. Örnek Politikalar:

It is permitted for actor(s) X to perform action(s) Y on target(s) Z.

It is forbidden for Client X to perform a communication action if the action has a destination of Chat Service Y.

Bir kullanıcı bir servis için istekte bulunduğunda, KAoS Grid servisi, politikaları kontrol eder ve eğer izinli ise, eylemi yürütmek için gerekli izinleri ve izin sürelerini içeren bir "GIS kısıt sertifikası" hazırlar. Kullanıcı, bu sertifika ile Grid servisine erişir.

[109] çalışması ise çok sayıda ve çeşitli tipte hesaplama kaynağı ve kompleks ortamların istek üzerine oluşturulması ve kaynağın tahsisi için otomatikleşmiş politika tabanlı bir model sunar. Politikalar, bu kaynakları oluştururken kullanılacak seçenekler üzerinde kısıtlar tanımlar (Şekil 4.1). Bu kısıtlar, kaynak tipleri içine gömülebilir, kaynak havuzunun operatörleri tarafından belirlenebilir veya isteğin bir parçası olarak kullanıcı tarafından belirtilebilir. Kaynak tipinden bir nesne oluşturulurken RMS, bu kaynak tipi ve alt hiyerarşideki bileşenleri ile ilişkili tüm kısıtların sağlandığından emin olur. Geleneksel PBM mimarilerinin aksine politikalarda bir eylem bölümü yoktur. Politikalarda yürütme anı dikkate alınmamıştır. Police modeline benzer olarak CIMOM tabanlı kaynak envanteri gerçekleştirilmiştir.

- Yeteneğe göre kaynak seçimi: Otomatik olarak uygun komponentler seçmek için, kullanıcının isteğinde ihtiyaç duyulan yetenekler belirtilir.
- Servis tipine göre kaynak seçimi: Genelde kullanıcı tipleri tanımlanır (platin, gold, gümüş vb) ve kullanıcılar bu sınıflara uygun olarak QoS garantisi verilir.

```

satisfy (grade == basic) ==> (server <: Server) && (server.grade == basic);
Proliant8500Server extends Server {
    satisfy grade \in {basic, advanced};
    // scaling up of ProliantServer
    satisfy numOfProcessors \in {1, 4, 8, 16};
    satisfy memtypes \in
    {onlineSpareMemory, HotPlugMirroredMemory, HotPlugRAIDMemory};
    satisfy powersupplyTypes \in {usual, redundantPowerSupply, UPS};
    // ileri düzey Proliant makinalar aşağıdaki yeteneklerdedir
    satisfy (grade == advanced) ==>
        (memtypes \in {HotPlugMirroredMemory, HotPlugRAIDMemory})
        &&
        (powerSupplyType <: UPS) && numOfProcessors == 16;
    // temel düzey Proliant makinalar aşağıdaki yeteneklerdedir
    satisfy (grade == basic) ==> (memtypes \in {onlineSpareMemory} ) &&
        ((powerSupplyType <: usual) ||
        (powerSupplyType <: redundantPowerSupply)) &&
        (numOfProcessors <= 8);
}

```

Şekil 4.1 : Kaynak inşaa politikası örneği.

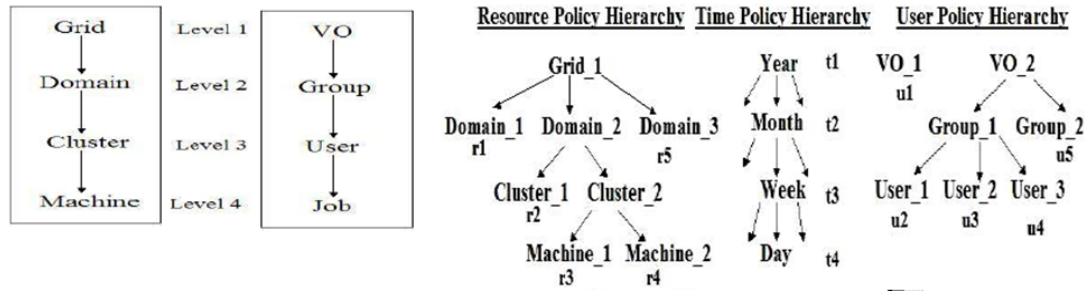
[110]'da, coğrafi olarak dağıtık, birbirine bağlı veri merkezleri olan ve çok sayıda uygulama arasında kaynak paylaşımı yapan büyük çaplı (Enterprise) Gridler üzerinde, Kaynak Topolojileri kavramı ve bu topolojileri tasarlamak için bir araç önerilmiştir. Kaynak tahsisleri ClassAds Eşleme yöntemi ile yapılır. Gridlerde istekler, bazı kısıtları sağlayan belli miktardaki hesaplama düğümleri için yapılır. İstekler, *RSL (Resource Specification Language)* dili kullanılarak <özellik,değer> çiftleri şeklinde belirtilir. Örnek: *[CPU=IA32, MEM>512MB, DISK> 20GB, OS=Linux]*. İstekler Grid Kaynak Tahsisi Yöneticisine (GRAM) gönderilir.

İhtiyaç duyulan kaynakların topolojisinin grafiksel olarak tasarlayan bir çalışma olan [111] makalesinde, kaynakların inşaaasında, önceki çalışmada tanımlanan politikalar kullanılmıştır. Kaynak istekleri daha üst seviye terimler olarak (Örneğin IA32 Linux sunucu) belirtilir ve bunlar fiziksel makine özelliklerine çevrilir. Bu araç ile sadece kaynakların statik bilgileri kullanılarak tanımlama yapılabilir. Bu tanımlama çevrimdışı olduğu için dinamik bilgileri kullanmaz. Kabaca yöntem, politikalara bakılarak havuzdan seçilecek kaynakları belirlemek ve kısıtlarla uyumlu kombinasyonları kullanarak kaynakları oluşturur. Kısıtlar, aritmetik ve lojik operatörler, veri tipler ve bazı anahtar kelimeler (for-all, exist, vb.) ile diğer yapılar (let in, if then else vb.) içerirler. Sistem operatörü tarafından belirlenen politikalar (sadece kısıtlardan oluşan basit politikalardır), komponent tiplerine bağlanırlar. Kullanıcının kaynak talebindeki

kısıtlar ile bu politikaların uyumluluğu kontrol edilir. Politika terimi, yine farklı anlamda kullanılmıştır. Bilinen PBM modeline uygun uygulama alt yapısı içermez.

QoS kavramı, bazı temel performans kriterleri ile açıklanabilir (performans, yanıt süresi ve başarımlar). [112] makalesinde QoS yönetimi konusu ele alınmıştır. Çeşitli servisler içeren şebekelerin yönetimi kompleks bir problemdir. SLA'lar, uygulamaların beklediği performans değerleri ile tanımlanır. Politikalar, istemciler, uygulamalar, sunucular, yaşam süresi, geçerlilik aralığı, çakışma çözme önceliği bilgileri kullanılarak oluşturulur. Politikalara uygun cihaz konfigürasyonları da oluşturulup cihazlara ulaştırılır.

[113] çalışması, SPHINX iş planlama sisteminin bir parçası olan, Gridlerde politika tabanlı iş sıralaması için kullanılacak bir altyapı tanıtır. Altyapı, hiyerarşik olarak 3 boyutta (kaynak sağlayıcılar, istekçiler ve zaman) politika tanımlamaya olanak sağlar (Şekil 4.2). Kaynak tahsisi için kullanıcının QoS isteğini ve kotaları dikkate alır. Tasarlanan sistem, politika tanımlaması, keşfi ve yorumlanması gibi PBM konularının aksine, iş sıralama için politika gerçekleştirme ve uygulama konularına odaklanmıştır. Heterojen sistemlerde, güncel durum bilgisini global olarak bilmenin zorluğu, kaynak tahsisine de hiyerarşik yaklaşmayı gerektirir. Bu nedenle politika tanımlama uzayı da çok hiyerarşilidir. *Kaynağı Sağlayan*, bir alt Grid veya bir *Gatekeeper* olabilir.



Şekil 4.2 : Önerilen politika tanımlama uzayları [113].

Genelde Lokal RMS'ler vardır (Condor, PBS, LSF). Politika yapısı [*Actor*, *Target*, *TimeConstraint*] şeklindedir (örneğin; $\langle u2, r2, t3 \rangle$). Politikalar tanımlandıktan sonra, isteklerin gereksinimlerini sağlayacak kaynak tahsisleri için optimal bir seçenek bulmada *Linear Programlama (LP)* kullanılır.

[114] çalışması, *Koalisyon* kavramı ile kaynakların paylaşıldığı bir modeli tanımlar. Kaynak paylaşımı, koalisyon üyeleri arasında imzalanan bir kontrat ile düzenlenir. Basit formatlı, *Koalisyon* ve *Lokal* adlı iki politika tipi vardır (Şekil 4.3):

- Koalisyon politikası (kontrat): Koalisyon üyelerinin arasında kaynakların paylaşımını düzenler.
 - Kontratlar bir dizi yükümlülükten (obligation) oluşur.
 - Bir kurum ancak kontratı kabul ederek kaynaklarını paylaşma şartlarını kabul ederse koalisyon üyesi olur. Örnek;
 - Kontratlar değişmez, sabittir.
- Lokal politika (plan): Bir üyenin hangi koşullarda, kaynaklarını diğer üyelerin kullanımına açtığını tanımlar.
 - Üyeler lokal politikalarını koalisyon içinde yayınlar.
 - Lokal politikalar koalisyon kontratı ile uyumlu olmalıdır. Ama bu uyumu kontrol eden bir otorite bulunmamaktadır.
 - Kurumlar lokal politikalarını değiştirebilir. Lokal politikasında tanımlasa da isterse kaynak tahsisinden vazgeçebilir.

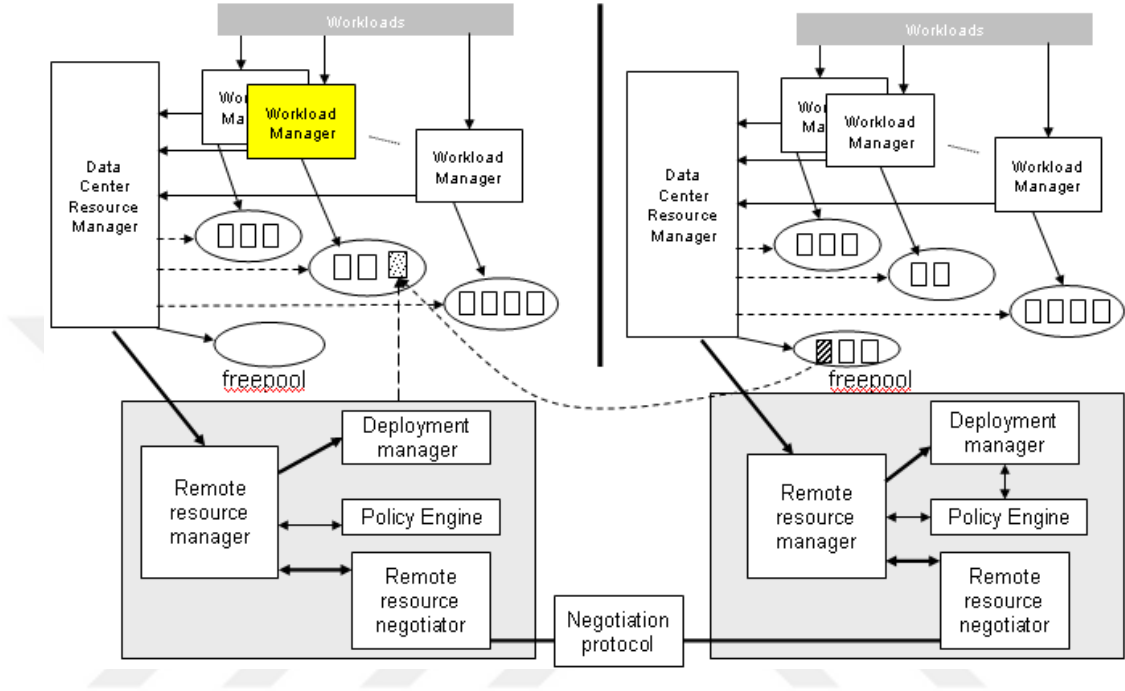
$Obl (P, S, R, I)$ • P: obligationın sahibi kurum • S: obligationla hak verilen kurumlar • R : kaynak ve miktarı • I: $[t_{start}, t_{end}]$ yükümlülüğün geçerli olduğu zaman aralığı	$Ent_A (E, R, I)$ • A: erişim için söz veren • E: yetki verilen kurumlar • R: söz verilen kaynak • I: zaman aralığı

Şekil 4.3 : Koalisyon ve Lokal politika formatları.

Bu modelin sorunu, yerel politikaların kontrat ile uyumluluk analiz gerektirmesidir. Ayrıca lokal politikalar değişebilirken kontratın değişmez olması da problemdir. Kontrata uyumu izlemek ve ihlalleri saptama ve düzeltme tanımlı değildir.

Grid operasyonlarında ağ alt yapısının yönetimi göz ardı edilmemelidir. Çünkü grid kaynaklarına erişim, ağ üzerinden olur. Grid sistemlerinde ise kullanılan ağ altyapısının yönetimi, gridin kaynak paylaşımı süreçlerinden bağımsız yapılıdır. [107]'de ise, gridle ağ altyapılarının entegre bir şekilde ve politika tabanlı olarak yönetilmesini amaçlamaktadır (Şekil 4.4). Grid politikaları, tanımlanan dönüşüm politikaları yardımıyla iletişim şebekesi politikalarına dönüştürülür. Dönüşüm kuralları her ağın kendi yöneticisi tarafından tanımlanır. Önerilen modelde, aslında iki katmanlı ve iki ayrı politika dilinden oluşan bir yapı vardır. Grid politika dili, iç içe (nested) kurallar tanımlamayı ve *domain* kavramını destekler. SPL [62] dilini, miras alma yeteneğini genişleterek kullanır. Grid politika dili ise IETF'nin kural modeline

[115] çalışmasında ise politika tabanlı olarak Veri Merkezleri arası dağıtık kaynak paylaşımı (kiralama) yapılır. Bağımsız Veri Merkezleri, kaynaklarını diğer Veri Merkezleri ile paylaşır. Her Veri Merkezi, kendi yönetim uzayına sahiptir (Şekil 4.6). Yerel RMS (*Workload Manager*), kendisine atanmış kaynaklar üzerinde her isteği tek başına ele alıp karar verir. RMS ise kaynakları WM'ler arasında paylaştırır.



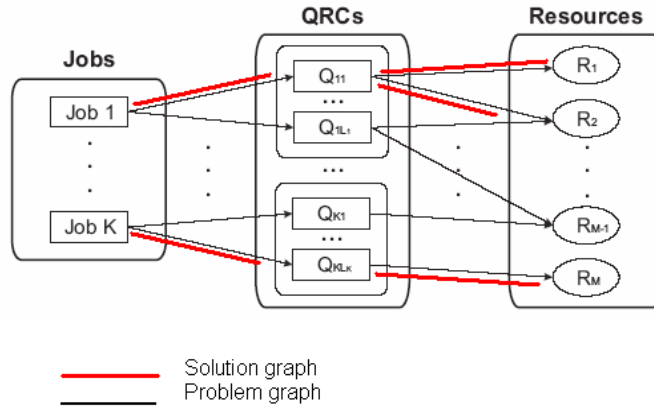
Şekil 4.6 : Veri Merkezleri için PBM modeli.

Kullanılan politika çeşitleri Şekil 4.7’de verilmiştir.

P1 – Uzak kaynak isteği politikası Condition workload id and priority Resource type Expected benefit Action Preferred datacenters //eğer varsa	P2 – Uzaktan kiralama politikası Condition Requester site id Requester workload id and priority Resource type Expected benefit Action Resource lending decision (qntity & duration)
P3 - İş dağıtımı vs. Kaynak ödünç alma Condition Workload id, Lending site id Action//how to use remote resources	P4 – Kaynakları geri alma politikası Condition Requester site id, Resource type Action Lease period
P5 – Kaynakları sağlama Condition Workload id, Lending site id, Requester site id Action – Provisioning site id	

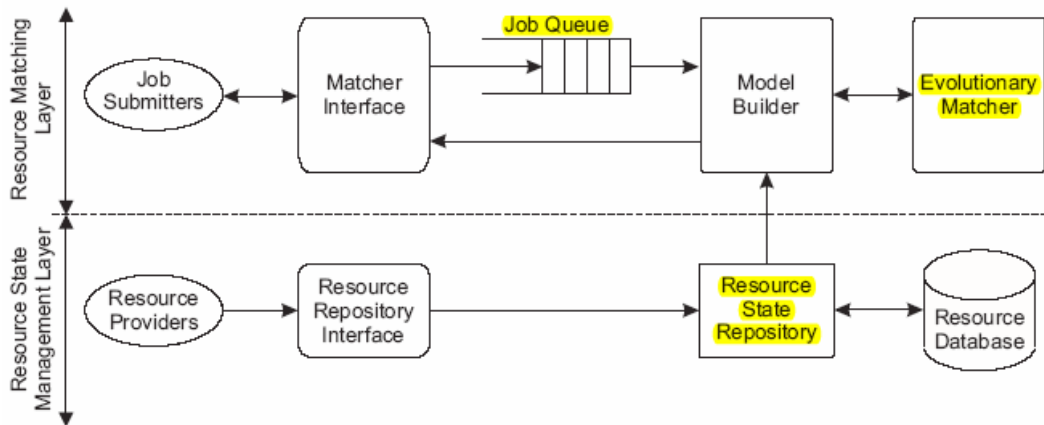
Şekil 4.7 : Veri Merkezleri arası kaynak paylaşım akışı ve politika çeşitleri.

[116] çalışması, heterojen kaynaklı, çoklu yönetimsel uzayları olan grid ortamlarında Kaynak Eşleme için geliştirilen sistemi tanıtır. Bu model kaynak eşleştirme için *EA* (*Evolutionary Algorithm*) kullanır ve dinamik kaynak paylaşımını, iş yüklerinin önceliğini, çoklu kaynak üzerindeki iş yüklerinin bağımlılıklarını ve kaynağa özel politikaları destekler. Varolan Grid servisleri ile entegre olabilir (Şekil 4.8). Bir iş yükü, kaynak özellikleri, zaman ve lokasyon kısıtları içerebilir. Benzer olarak kaynakların da çalıştırmak iş yükü tipleri ve zaman tercihleri olabilir (*resource policy*). Site yöneticileri global kaynak kullanımı politikaları tanımlayabilir. Ama politika ile kastedilen, kaynak eşleyicinin önceden tanımlı bir kaç çalışma modudur ve gerçek anlamda politika değildir. Bu çalışma modlarına örnek olarak, *maximize throughput*, *maximize prioritized throughput* ve *load balance* verilebilir. Bu politikalar, kaynak eşleştirmede, *objektif* fonksiyonu veya *fitness* değeri olarak modellenir (Şekil 4.9).



Şekil 4.8 : İş yükü bağımlılıkları için graf ve kaynak eşleme örneği [116].

Qualifying Resource Collection (QRC), bir iş yükünün ihtiyaç duyduğu minimum kaynak kümesidir (kapasite ve süre). Evolutionary Algoritmalarını (EA) optimizasyon

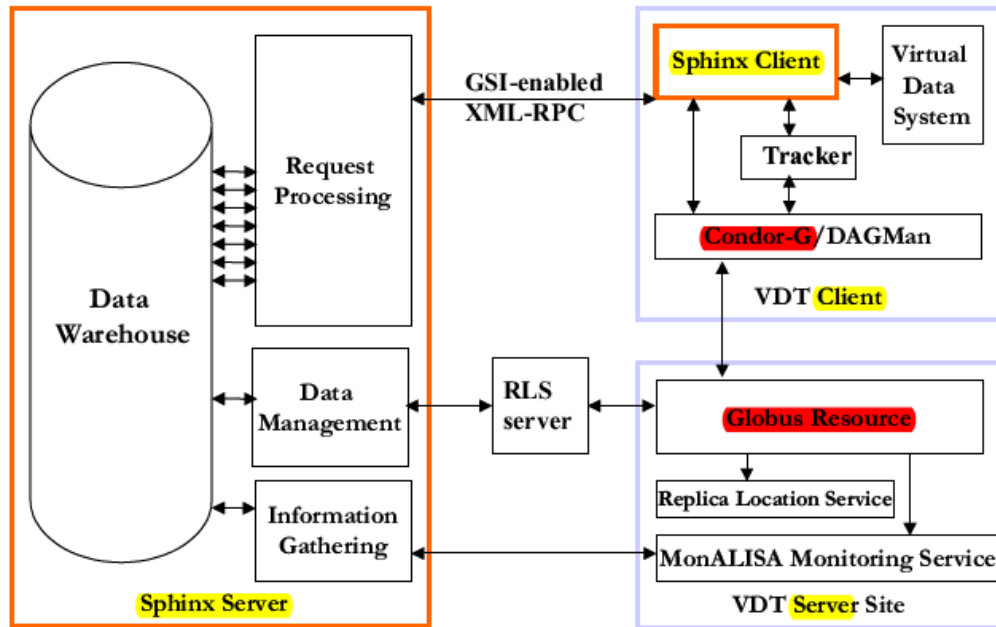


Şekil 4.9 : Önerilen Kaynak Eşleyici (RM) modeli.

yöntemi olarak kullanır. EA'lar çözüm seçenekleri üzerinde rasgele arama yapar ve heterojen gridlerde kaynak eşleme arar (EM'nin girdileri, bir dizi iş yükü için kaynak gereksinimleri, öncelikleri, tercihleri, şimdiki kaynak durumu ve kaynak/site politikalarıdır). Önce bir Başlangıç Çözüm ataması yapılır. Sonra iteratif olarak belli kriterler sağlanıncaya kadar çözüm aranır. Bu çözümlerin *fitness değerlerine* göre bir çözüm kümesi seçilir.

Çoklu kriter kullanarak iş yüklerini kaynaklara atayan *Genetik Super-Scheduler* çok fazla zaman tükettiğinden on-line kaynak eşlemeye uygun değildir.

[117]'da, SPHINX altyapısında (Şekil 4.10), sitelere gönderilmiş eski iş yüklerinin istatistiğine bakıp hata oluşan bilgisayarlara daha az iş yükü gönderilir. SPHINX altyapısı, çok sayıda ve farklı kurumlara ait dağıtık cluster üzerinde, farklı lokal



Şekil 4.10 : SPHINX mimarisi [117].

iş sıralama politikaları varken çalışabilir. Heterojen kaynakları izleyerek insan müdahalesi gerekmeden iş sıralaması yapar. SPHINX, kaynak politika kısıtlarını destekler. Hem harici gözlem sistemlerini hem de kendi izlediği iş yükü tamamlanma süre bilgilerini kullanabilir. *Sphinx Client*, kullanıcı isteklerini kabul edip sunucuya iletir. Sphinx Server, Sphinx Client'ın gönderdiği iş yükü bilgisine göre işi sıraya koyar. Gözlem bilgilerinden ve RLS (Replika Location Service)'ten faydalanan Client, iş yükünü, sunucunun belirttiği siteye gönderir. *Tracker* ise gönderilmiş iş yüklerinin yürütme durumunun değişimini izler ve gözlem bilgisini sunucuya iletir. Sphinx

Sunucu, bu kaynaklar üzerinde işin tamamlanma süresini tahmin eder. SPHINX iş planlama servisi, bazı grid servisleri ile etkileşir (Globus, Condor_q ve PBS).

Girdi verileri hazır olan bir iş yükü kümesi seçilir; her bir iş yükü için optimum kaynak ataması yapılır (bu atama, kaynakların durumları, kullanım politikaları, iş yükü yürütme kestirimi ve I/O bağımlılıklarına göre yapılır); gerekiyorsa yürütücü siteye girdi dosyaları transfer edilir. Çalışırken, Grid sitelerine önceki gönderilmiş iş yüklerinin yürütme durum bilgilerini ve Grid bileşenlerinin gözlenen performans bilgisini kullanır.

En çok referans verilen çalışmalardan birisi olan [118], *Kullanım Politikası (Usage Policy)* tabanlı grid kaynak planlamasını, Lokal ve VO (Virtual Organization) politikalarının tanımlanması, yorumlanması ve uygulanmasını açıklar. Yazarlar, gridlerde kullanılmak üzere bir politika yönetimi mimarisi önermişlerdir. Kaynak sahipleri (site, VO vb.) ile kullanıcılar arasındaki ilişki bir SLA'ya tabidir. SLA'lar tanımlanabilir, keşfedilebilir, yorumlanabilir, doğrulanabilir, pazarlık edilebilir ve uygulanabilir. Politikaların uygulanması için İki tip PEP gerekir:

- Site PEP (S-PEP): Tüm sitelerde siteye özel politikaları uygular. Sürekli çalışır ve politikalara uymayan işler derhal sonlandırır. LS'lerden iş yüklerinin durum bilgilerini alır. CPU kullanımını hesaplar, LS'ye işlerin start/stop/preempt emirlerini verir. S-PEP gerçeklemek için iki seçenek önerilmiş:

1-Standalone S-PEP: SPEP, LS'lerin yanında bulunur ve politikalara göre LS'yi yönetir. S-PEP aşağıdakileri yapabilen herhangi bir-batch- RMS sistemi ile çalışabilir:

- İşler hakkında kullanım ve durum bilgilerini sağlayabilen
- İş yükü başlatma/durdurma/tutma/silme yetenekleri olan
- Çalışan işlerin önceliğini değiştirebilen

S-PEP ya öncelik tabanlı (İşlerin önceliğiyle oynayarak) veya uygunluk tabanlı (Tüm işler aynı öncelikte. İş yüklerini beklet, başlat, preempt yaparak) politika uygular. Bu PEP modeli, işlemci, disk ve şebeke dışındaki kaynak tiplerine destek sağlayamaz.

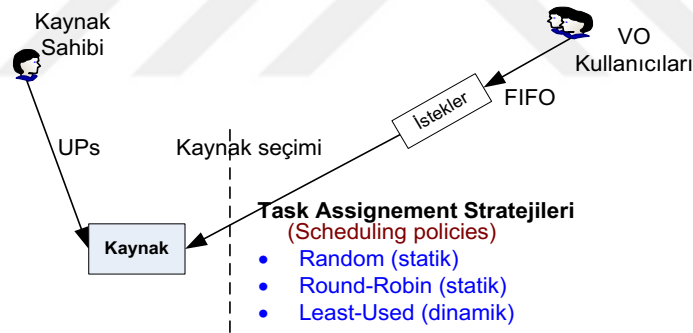
2- RMS üzerine S-PEP (Politika tabanlı) : Grid3 üzerinde gerçekleşir. S-PEP fonksiyonları, S-POP (Standalone Site Policy Observation Point) ve Cluster RMS (Condor, PBS, LSF vb.) arasında paylaşılır. Küme RMS'in, S-POP

tarafından sağlanan UP'leri yürütebildiği varsayılır. S-POP'un görevi, UP'leri RMS'in anlayacağı hale çevirmek ve gerçek kaynak kullanımıdır. Aynı S-POP birden fazla RMS ile birlikte çalışabilir.

- VO PEP (V-PEPs): S-PEP'e benzer şekilde çalışır. İş yükü başına karar verir. VO gruplarına kaynak tahsisinde, VO'nun kendi politikalarını uygular. VO iş yükleri için planlama kararı verilirken V-PEP tetiklenir. V-PEP, VO politikalarını uygulamak için S-PEP'ler ve iş sıralayıcılar ile etkileşir.

Yazarlar, PEP'lerde gözlem desteği için *Ganglia Cluster Monitor* adlı aracı kullanmışlardır. *MDP (Monitoring Distribution Point)* adlı modüller, hem gözlem hem de politika dağıtımı yapar. Kaynakların VO kullanıcılarına tahsisinde, basit formatlı UP (Usage Policy)'ler kullanılır (Şekil 4.11 ve Şekil 4.12). Diğer alanlardaki (erişim denetimi, kaynak yönetimi, vb.) politikalar yoktur.

Aynı yazarların *UP (Usage Policy)* tabanlı CPU paylaşımı çalışması [119], pek çok VO'lu Grid3 ortamlarına hangi UP konfigürasyonunun daha uygun olduğunu ve VO iş sıralayıcılar için UP'lerin nasıl kullanılabileceğini irdeler.



Şekil 4.11 : VO kullanıcıları ve kaynak sahiplerinin ilişkisi.

Hiyerarşik politika tanımlanabilir. Kaynak sahipleri VO'lara haklar vermek için UP tanımlar. VO'lar ayrıca, (kaynak sahiplerinden) topladıkları kaynakların yönetimi için kendi politikalarını da tanımlayabilirler. Site seçimi için Şekil 4.11'deki üç çeşit stratejiyi önerirler. UP formatı, gerçek anlamda bir yönetim politikasından çok, Bölüm 3.1.3.2'de tanıtilen düşük seviye kural tabanlı ağ politikalarına benzer. Dolayısıyla sınırlı işleve sahiptir. Yazarların önerdikleri sistemin mimari modelleri Şekil 4.13 ve Şekil 4.14'de verilmiştir.

//UP formatı

<resource-type, provider(sites), consumer(VOs), epoch-allocation, burst-allocation >
epoch/burst-allocation ::= (interval, percentage)

//Örnek

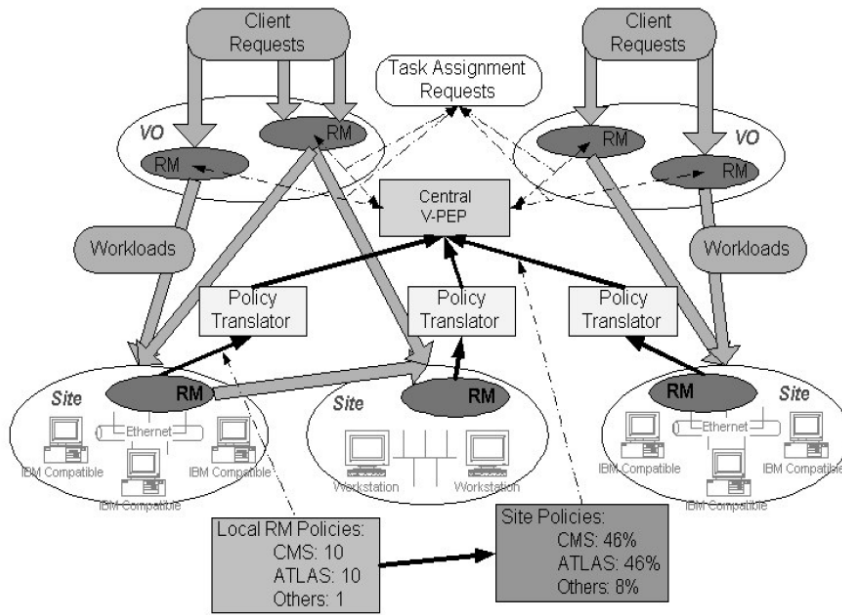
[CPU, Site1, VO0, (3600sn, 10%), (60sn, 20%)]

T_{epoch} , R_{epoch} T_{burst} R_{burst}

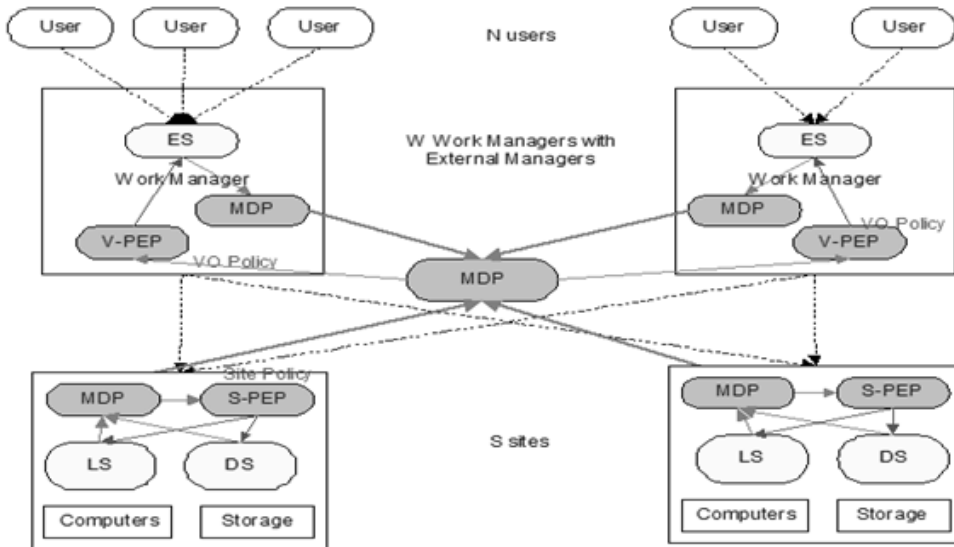
İşler 1 saat süresince %10 u geçemez. Bir dakika içerisinde de %20'ye çıkabilir.
Ama kısa bir periyotta (10 sn gibi) %100'e fırlayabilir.

Şekil 4.12 : UP formatı.

Yazarlar önerdikleri sistemin etkinliğini ölçmek için hem simülasyon çalışması hem de gerçek Grid ortamında (Grid3) deneysel çalışmalar yapmışlardır [120].



Şekil 4.13 : VO tabanlı HPC modeli [120].



Şekil 4.14 : GangSim'in bileşenleri [120].

ES (External Scheduler), kullanıcı işlerini kuyruklar ve her iş için en iyi aday siteyi seçer. İş yükü, ES'nin seçtiği sitedeki LS (Lokal Scheduler)'a gelir. MDP'ler (Monitoring Distribution Points), çeşitli metrikleri (ES ve LS'den) ölçen gözlem düğümleridir. Aynı zamanda politikaların alt MDP'lere dağıtımında da rol oynarlar. PEP'ler politikaları kaynak tahsisinde kullanır. Bir iş yükü LS'de reddedildiğinde, tekrar planlanmak üzere ES kuyruğuna geri döner.

[113]'de ise yukarıdaki çalışmadan farklı olarak Merkezi PEP kullanılmıştır. Bazı başka yazarlar kaynakları paylaşmak yerine, servis olarak pazarlamayı önermişlerdir (Öde ve Kullar). Bu modelde politikalar, kazanan istekçileri belirlemek ve kaynakları buna uygun planlamak için tanımlanır.

[104] çalışması ise Data Gridlerinde iş yüklerinin girdi dosyalarının kullanım stratejilerini konu alır. Bu çalışmada da yazarlar sistemin çalışma yöntemine politika adını vererek politika kavramını yanlış kullanmıştır.

[121] çalışmasında, data gridlerinde işlerin dosya kullanımını politikalarla yönetmek amaçlanmıştır. Önerilen modelde, işler kuyruklanır ve bir sonraki adımda hangi işe ve bu işin hangi dosya isteğine hizmet vermesi gerektiğine karar verilir. Kullanılan politikalar türleri şunlardır:

- Servis Politikaları: Bir sonraki işi seçmek için lokal politikalar ile belirlenmiş kısıtlardır (bir iş yükünün kullanacağı disk alanı, aynı anlık dosya sayısı, vb).
- Dosya (File Caching) Politikaları: Seçilen işe ait hangi dosyanın, disk cebine alınması gerektiği belirlenir.
- Cepten Atma (Cache Replacement) Politikaları: Yer kalmadığında hangi dosyanın cepten silineceğini tanımlar.
- Kopyalama (Replication) Politikası: Hangi dosyanın ne zaman, nereye ve kaç kopya gönderileceğini belirler.

Bu makalede de politika kelimesi yanlış anlamda kullanılarak statik konfigürasyonlara (strateji) politika denilmiştir. Örneğin, servis politikası olarak *OptCacheLoad* (Hizmet verilen iş yüklerinin toplam değerini maksimize eden dosya alt kümesini cebe yüklemek), Cepten çıkarma politikası olarak *LRU (Least Recently Used)* uygulamak, politikadan ziyade birer çalışma modu parametresine karşı düşer.

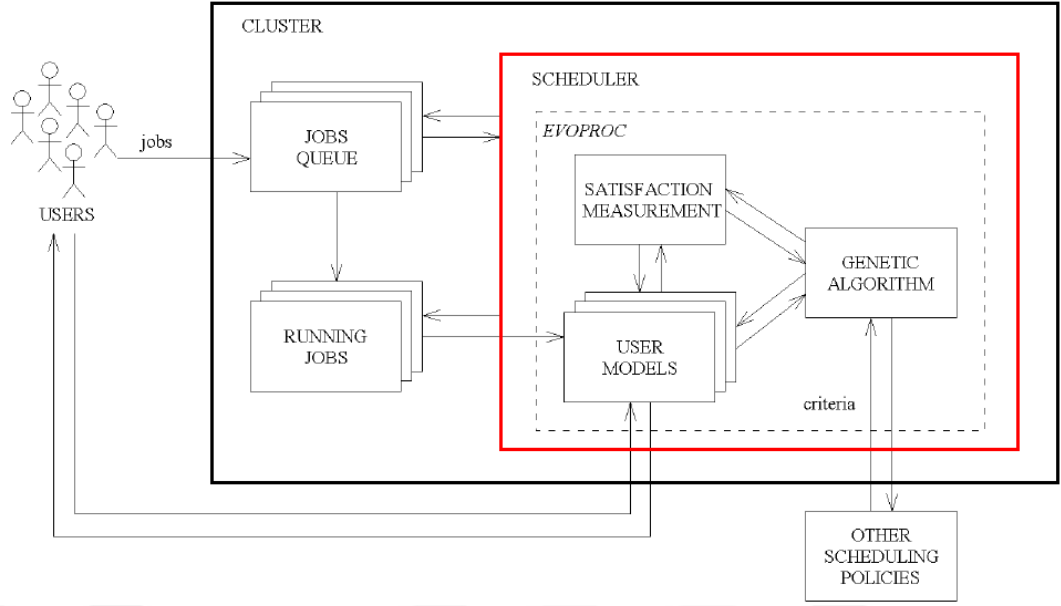
[122] çalışması, Genetik Algoritmalar kullanarak iş sıralama kalitesini artırmaya çalışır. Bu çalışmada bazı örnek kısıtlar (iş yükü mimarisi, gerekli hesaplama gücü)

seçilip iş planlamayı optimize etmek için evrimsel teknikler kullanılır. Sıralamada, iş yükü, Grid ortamı ve veri dağıtım karakteristikleri gibi statik veriler kullanılır. Uygunluk (fitness) fonksiyonu olarak, her bir paralel bilgisayar başına ağırlıklı iş bitirme (weighted makespan) değeri hesaplanır ve böylece grid düğümlerinde bir yük dengesi sağlanır. Grid düğümleri, politika olarak yanlış isimlendirilmiş olan FCFS vb. stratejilere göre lokal kaynaklarını yönetir.

[123] makalesinde, bir interaktif grid örneği tanıtılmıştır. Gridlere gönderilen kullanıcı etkileşimi gerektiren bu tür uygulamalar, çok uzun süre yaşar, daha çok ve eş zamanlı kullanıcı isteği oluşur. Sistem, her istemci isteğini alır ve bir politikaya göre sanal servis düğümüne iletir. Yine, politika adı verilen, *weighted round-robin* aslında politika değil bir çalışma modudur.

[124] çalışmasında, kaynakların konfigürasyonu ve şimdiki durum bilgilerini bilmeye gerek bırakmayan bir çözüm önerilmiştir. İş yükü yürütme düğümleri, koordinatör düğümden işleri çekip yürütürler. Yürütücü düğümlerden gelen isteklere göre iş bitirme hızları hesaplanır. Böylece iş yükü, düğümlere sormadan atanabilir. Yazarların politika adını verdikleri çalışma modları vardır: *CRSS (CR self-scheduling)*, *CSS (Chunk Self-Scheduling)* ve *WF (Weighted Applied Movable Master Node Factoring)*. Önceden deneylerle belirlenmiş sabit eşikler kullanması sebebiyle değişken gridlerde yeterli verimi sağlamaz.

[125] çalışmasında, konfigürasyon zorluğu, kullanıcıların işlerin ihtiyaçlarını doğru belirtememesi, sistem performansının her kullanıcı tarafından eşit farkedilir olmaması, işlerin beklenenden daha erken ölebilmesi problemlerini adreslemek için yazarlar, üç önemli bileşen kullanan bir sistem önermiştir: genetik algoritmalar, kullanıcı davranış modeli ve kullanıcı tatmin modeli. Bir genetik algoritma, işlerin yürütme sırası için en iyi sıralamayı bulmaya çalışır. Bu amaçla, her bir kullanıcı için otomatik olarak elde edilen ve işler için gereken gerçek kaynak miktarını tahmin eden bir kullanıcı davranış modeli ve kullanıcı tatminini tahmin etmek için kullanıcı tatmin modeli (*user satisfaction model*) ile çalışan bir uyum (fitness) fonksiyonu kullanırlar. Sistem kullanıcı davranışını öğrenir ve sistem kaynaklarının kullanıcılarca gerçek kullanımını tahmin edebilir. Modellerini (EVOPROC) Sun Grid Engine (SGE) içine entegre etmişlerdir (Şekil 4.15). İş sıralayıcı, bir genetik algoritma kullanır.



Şekil 4.15 : Önerilen model [125].

İş sıralayıcı için 6 adet basit uygunluk fonksiyonu gerçekleştirilmiştir:

- Minimize makespan (En son iş yükü en kısa zamanda tamamlanmalı)
- Minimize maximum waiting time
- Minimize average waiting time
- Minimize waiting time per CPU unit time (Tüm işler kendi yürütme süreleri ile orantılı olarak beklemeli)
- Maximize minimum satisfaction
- Maximize average satisfaction

Bu uygunluk fonksiyonları (stratejiler), yanlış bir şekilde, politikalar olarak adlandırılmıştır. Bu sistemde gerçek anlamda bir politika dili yoktur.

[126] çalışmasında, *kaynak ticareti politikaları* kullanılmıştır. Kaynakların adil paylaşıldığından emin olmak için her kuruma katkıları ile orantılı sayıda jeton verilir. Muhasebe parametreleri kullanarak rekabet izlenir. Her bir etmende tanımlanan politikalar, bu muhasebe parametrelerinin değerlerine göre, talep ve arz fiyat indekslerini ayarlayabilir. Kullanıcılar isteklerini kendi kurumlarının CA'ları (Consumer Agentlar) üzerinden gönderir. Her kurumun site iş sıralayıcısı vardır ve kabul edilen isteklerin tahsisini yapar. Her bir PA (Provider Agent), kaynaklarına erişim fiyatını jeton cinsinden belirler. BA (Broker Agent), istekleri kuyruklar ve tüm

kurumların PA'larına iletir. Her PA, servisin ücretini BA'ya bildirir. BA, kullanılacak PA'yı seçer. Bu çalışmada fiyat katsayılarını belirlemek için *if else-if* formatında *ticaret politikaları* kullanılmıştır.

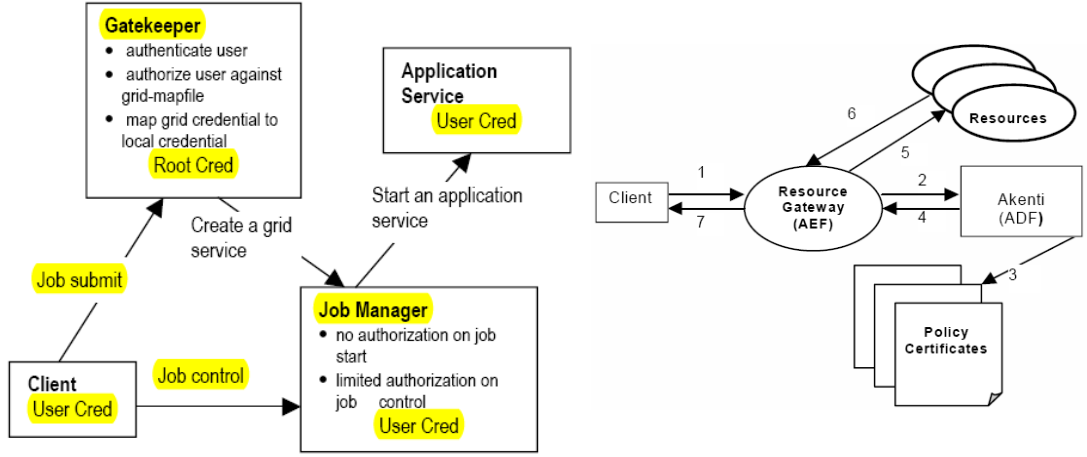
İş yükü isteklerinin sayısı yüksekse ve kümenin durum değişiklikleri sık oluyorsa, iş yüklerini toplu planlamak, tek tek planlamaktan daha iyi sonuç verir [24]. Yazarlar, Bulanık Mantık (Fuzzy) tabanlı bir iş *sıralama politikası* önermişlerdir. Bu çalışmada da politika kelimesi, kaynak eşleme süreçlerindeki stratejiler için kullanılmıştır. Yazarlar *Match-making* ve Kuyruk kavramını birlikte kullanırlar. 'ClassAd' iş sıralama yönteminin, işleri tek tek işlemek, kaynak sahiplerinin kısıtlarını dikkate almamak, önceden iş yükü bilgisi gerektirmek gibi dezavantajları vardır. Bunları gidermek için, iş yükleri bir kuyrukta bir süre (T) bekletilerek birlikte planlanır. Kaynak sahipleri, paylaşım açacakları kaynaklarını, kısıtları ile birlikte ilan ederler. Kullanıcılar ise iş yüklerini, gereksinimlerini RSL'de ifade ederek gönderirler.

Literatürdeki çalışmaların çoğu, kaynak tahsisi algoritmaları konusunda olsa da Güvenlik ve Erişim Denetimi konusunda da çalışmalar vardır. Bu çalışmalarda, kullanıcı isteklerini, kaynaklara ait detaylı (fine-grained) erişim denetim politikaları ile karşılamak amaçlanır.

[127]'de, kaynak ve görevlerin yönetiminde yetki politikaları kullanan *SAS (Sub-task-based Authorization Service)* mimarisi tanıtılmıştır. Kimlik doğrulama, yetkilendirme ve erişim denetimi, hem kaynaklara hem de iş yüklerine uygulanır. Kullanıcının iş yükü yönetimi istekleri ve iş yükünün kaynak talebi, kaynak sahibi tarafından tanımlanmış politikalar ile değerlendirilmiş olur.

[128] çalışması, *Globus Toolkit 2* ve *Akenti Yetki Servisi* kullanan sistemlerde, iş yükü yönetimi için *politika tabanlı yetkilendirme* çözümü sunar. Kullanıcının RSL dilinde belirttiği iş yükü istekleri, sitelerin yetki politikalarına göre değerlendirilir. *FGAC (FineGrained Access Control)* için, erişim denetimi karar fonksiyonu (*ADF*), politikalara göre erişim izinlerini verir.

Globus Toolkit'de Yetkilendirme: GT2, detaylı olmayan bir giriş kontrolü için kaynak sahiplerine özel FGAC ekleme olanağı sunar. Kullanıcı, iş yükü isteğini (iş yükü tanımı + kullanıcının Grid doğrulaması sertifikası) gönderir. İstek, kaynak yönetimi modüllerindeki Erişim Denetimi Modülü (*AEFs*)'nin erişim denetimi karar fonksiyonu (*ADF*) tarafından değerlendirildikten sonra başlatılır (Şekil 4.16).



Şekil 4.16 : Globus Şemasının Akenti Sunucu ile entegrasyonu [128].

GRAM Gatekeeper, istekçi kullanıcısının kimliğini GSI üzerinden doğrular, yapılacak işlemin yetki kontrolünü yapar ve iş yükünün hangi hesap altında çalıştırılacağını belirler. *Job Manager*, istekçinin Grid kimlik bilgisini yerel kimlik bilgisine çevirir ve iş yürütmeye ilgili istekleri kotarır. Kullanıcının, kaynak üzerinde bir hesabı bulunmalıdır. *Job Manager*, bir iş yükünü başlatır ve yönetir. Kaynağın ait olduğu JMS sistemiyle (LSF, PBS vb.) etkileşir. JMS, kullanıcıya kendi üzerindeki politikaları uygular. GT2'ye ekledikleri Akenti, dağıtık kaynaklara erişim kararlarını vermek için kullanılan bir yetkilendirme servisi. Kullanıcıların X.509 sertifikaları vardır. Bir kaynak için sahibinin tanımladığı Yetkilendirme politikası, aslında bir dizi sertifikadır. Bu sertifikalar, erişecek kullanıcıların sahip olması gereken özellikleri tanımlar. Kaynağın bekçisi (AEF), güvenilir bir Akenti Sunucuya (ADF), kullanıcının kaynağa erişim yetkisini sorar. Akenti Sunucu, sertifikaları doğrulayarak cevap döner. Akenti politikası [128] XML olarak ifade edilir ve 3 tip imzalı sertifika vardır:

- Politika sertifikası (kaynağın otoritelerini belirtir)
- Kullanım koşulu sertifikası (kaynağa erişim kısıtlarını içerir)
- Özellik sertifikası (kullanıcı özellikleri tanımlar)

Sertifikaların DN (Distinguished Name) alanında, rol, grup ve eğitim seviyesi gibi *AKENTI* özellikleri yanı sıra zaman, makina yük durumu gibi *SYSTEM* özellikleri bulunur. Örnek politika aşağıdaki gibidir:

```
(DN=/O=DOEGrids/OU=People/CN=Jane Doe) || (role=developer &&
(time>5pm) && (time<8am)) || (group=clients && executable=TRANSP)
actions=start
```

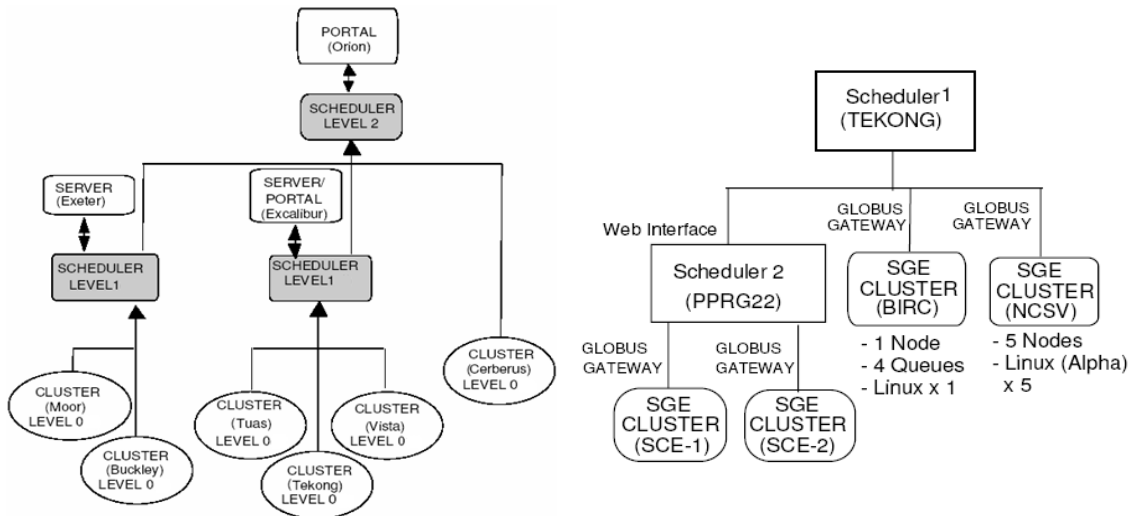
Akenti iş yükü yönetim isteklerini erişim kontrolünden geçirir. Politikalar kaynağın bulunduğu bilgisayarda depolanır. Politikalar arasında çakışma çözme ve politika yaşam çevrimini yönetme gibi gerçek bir PBM’de olması gereken yetenekler yoktur.

İş sıralayıcılar, erişim denetimi ile koordinasyon içinde iş yükü sıralama yapmalıdır. [129]’de yazarların önerdikleri politika tabanlı erişim modelinde, iş sıralayıcı içindeki güvenlik bileşeni, tahsis öncesinde kaynakların politikalarını (kısıtları) değerlendirerek kullanıcının erişim yetkisinin olduğu kaynaklara atama yapar. Yetki Politikaları, çok basit kısıtlardan oluşur ve IETF formatına yakındır:

Target(erişim istekleri kümesi), Effect(deny/accept), Condition

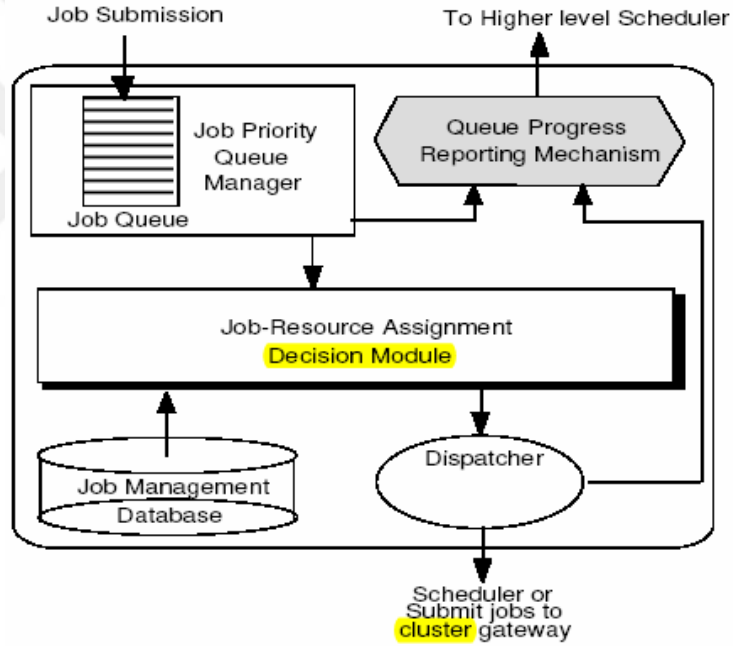
Merkezi Politika Yöneticisi, Gride katılan düğümlerin politikalarını merkeze toplar (global politikalarla tutarlılığını kontrol eder) ve politikaların benzerliğine göre kaynakları bölmeler. Bu yöntem, politikaların çok sık değiştiği senaryolara uygun değildir. Sadece kaynakların özelliklerine göre politikalarla eşleme (match-making) yapılıyorsa uygundur. Politika uygulama yapısı esnek değildir, genişleyemez. Politikaların içinde, kullanıcıların sağlayabildiğinden başka değişkenler kullanılamaz.

[130], iş yükü yönetimi için, dağıtılmış hiyerarşik iş sıralayıcı şeması önerir. Kaynaklarını paylaşım açanlar kaynak havuzuna erişim hakkı elde ederler. *Clearinghouse* iş sıralayıcı, iş yükünü bir kümeye veya QoS gereklerini en iyi sağlayabilecek alt seviye iş sıralayıcıya gönderir. Kullanıcı, iş yükü çalıştırmak için kaç kredi harcayabileceğini belirterek (ücret) iş yükünü gönderir. İş tamamlanınca yürüten site, iş sahibinin global kredi hesabından tahsil eder (Şekil 4.17).



Şekil 4.17 : Kullanılan iki seviyeli iş sıralama şeması [130].

Clearinghouse'a kayıt olan kurumlar VO oluşturmak üzere hiyerarşilendirilir. Bir çok küme ve Globus Toolkit ile inşa edilmiş iki hiyerarşili kampüs grid kullanılmıştır. Her bir küme için bir Globus gatekeeper vardır ve kümelere gönderilen iş yükleri için bir geçit görevi görür. İlk seviye (Level₀), her biri farklı RMS tarafından yönetilen, heterojen düğümlerden oluşan kümelerdir. RMS, düğümlerin durum bilgisini daha üst seviyedeki iş sıralayıcılara iletir. Kümeler tek başlarına çalışırlar veya iş yüklerini kendi lokal iş sıralayıcıları ile yönetecekleri daha geniş guruplar (Level₁) oluştururlar. Hiyerarik olan iş sıralayıcılar dağıttıktır ama ödeme işlemleri merkezi olarak yapılır. Level₁'e iş yükü gelince dağıtmak için kendi kümelerinden birisini seçer. Level₂ iş sıralayıcı ise gelen iş yükünü ya Level₁ iş sıralayıcılara veya doğrudan bağımsız bir Level₀ kümeye gönderir. Sistemin iş sıralama akışı Şekil 4.18'de gösterilmiştir.



Şekil 4.18 : İş sıralama düğümü yapısı [130].

İş yükü önceliği, iş yükü ile birlikte sağlanmış olan kredi (fiyat önerisi) değerlerine ve kuyrukta bekleme süresine dayanır. Karar modulünde, hem *politika parametreleri* hem de kümelerden gelen yük bilgileri kullanılır.

Tüm performans parametrelerinin (lokal yöneticilerin politikaları ve alt hiyerarşideki kümelerden gelen bilgiler) ağırlıklı toplamı kullanılarak seçim için kümeler sıraya konulur. Bu bilgilerle, küme matrisi oluşturulur. Ağırlıklandırılmış elemanlardan oluşan bu matris aşağıdaki parametrelerden oluşur:

- Nominal Hesaplama gücü.
- Uygulama Tercihi. Kümelere kullanıcının belirlediği tercihlere göre öncelik vermeye yarar
- Küme Denetim Politikası. Yöneticilerin kendi kümeleri için tanımladıkları politikalar (RR, LRU Cluster, vb).
- Küme Kredi Bakiyesi

Yazarlar bu parametreleri ve çalışma modlarını Politika diye adlandırmış olsalar da aslında bunlar sadece kısıtlardır. Ağırlık katsayıları sabittir ve elle ayarlanır.

Çoğu araştırma, iş yüklerini biriktirip çalıştırma (batch jobs) odaklıdır. Ama gelecekteki gridlerde interaktif GUI oturumlar da kullanılacak. Interaktif gridler, yeni problem alanları olan, FGAC, QoS desteği ve Dinamik hesap yönetimi konularında çözüm sağlar.

I-GENV ([131]), interaktif grid oluşturmak için Globus Toolkit 2.0'ı genişleterek uzak düğümlere grafiksel işler de gönderilebilme olanağı sunan bir ortamdır. Çeşitli tipte politikalar kullanır. Politikalarla yönetilen FGAC yeteneğindeki bir erişim denetimi sistemi vardır. Tüm multimedya işler *GDRM (Grid Distributed Resource Management)* sistemine gönderilir. İsteğin parametrelerine göre uzak bir düğüm seçilir ve oturum boyunca rezerve edilir.

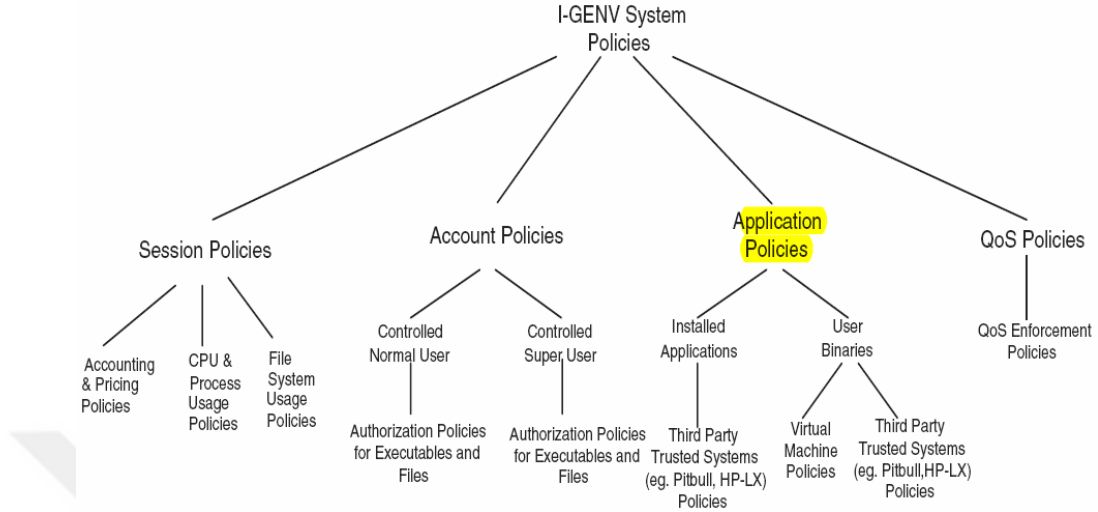
SAC (Session Admission Control) sistemi, Gride giriş denetimi kararlarını, SLA ve oturum politikalarına göre verir. Karar sürecinin girdileri: SLA, Uygulama ve Profili, Politikalar, GMMA'ların topladığı kaynak kullanım durumu verileridir. SAC, bir oturumda SLA ihlali oluşursa, SLA Uygulama politikalarında belirtilen eylemleri yürütür (mesela süreçleri öldür veya oturumu kapat vb).

Yazarların kullanılan politika türleri (Şekil 4.19) aşağıdakilerdir:

- Oturum (Session) Politikaları: Her bir oturum başına sistem parametrelerinin ihlal durumları için öntanımlı eylemleri belirler.
- Hesap (Account) Politikaları: Normal ve süper kullanıcılar için ayrı politikalar.
- Uygulama (Application) Politikaları: I-GENV tarafından başlatılan uygulamalar için politikalar
- QoS Politikaları: QoS metrikleri ve uygulaması için

Uygulama Profiliinde belirtilen, tahmini CPU ve ağ bantgenişliği parametrelerinin ihlal edilip edilmediği (QS) yönetilir. Erişim Denetim Modülü, yürütülecek uygulamanın,

izinli listesinde olduğundan, izinli argümanlarla çağrıldığından ve sadece izinli dosya/dizinlere eriştiğinden emin olur. Politika dosyasında bu maddelerin uygulanması için bilgi vardır.



Şekil 4.19 : Kullanılan politika türleri [131].

Sistemin politika altyapısı, esnek bir PBM değildir ve dili basittir. Grid kullanıcıları, VO politikasına göre, sistem politika dosyaları ile ilişkilendirilmiş olan dinamik kullanıcı hesap havuzlarından birisine atanır ve bu havuza ait politikalara tabi olur. Fakat bu yapı esnek değildir. Zira kullanıcı hesaplarına, önceden tanımlanmış yetkiler verilir. Uygulamalar için de global profiller tanımlanmıştır ve bunlara dayalı QoS planlaması yapılır. Daha esnek yetki/politika tanımlanabilmesi gerekir.

[132] makalesinde, toplu (batch) iş yüklerinin, interaktif kullanımlardan etkilenmesini ve kaynakların atıl kalmasını önlemek amaçlanarak dağıtık kuyrukların (DQM) politikalarla yönetilmesi konusu vurgulanmıştır. Batch iş yükü yürütmek için politika tanımlanır ve yönetim sistemine uygun komutlara dönüştürülür. HPOpenView yazılımı kullanarak DQM'leri yönetir.

[133] makalesi, kaynakların istemciler tarafından arz-talep prensibine dayanan kullanımı için ekonomi tabanlı iş ve kaynak yönetimi modeli önerisi sunar. Var olan Grid ortamı kaynak tahsisi yaklaşımları gevşek bağlıdır. En üst seviye iş sıralayıcılar (Application Scheduler), uygulamaları, anlık yük durumunu, öncelikleri, diğer uygulama iş sıralayıcıların kullanım senaryolarını dikkate almadan gönderirler. Aralarında koordinasyon olmadığı için sonuçta kaynaklarda yük dengesizliği oluşur.

Grid kaynakları, Condor, PBS gibi Lokal RMS ile yönetilir. İş yükleri Lokal RMS'ye, cevap süresi veya servisin kullanılabilirliği hakkında bilgi sahibi olunmadan gönderilir. İşler saatlerce kuyrukta bekleyebilir.

İş planlayıcılar ya cevap süresini minimize etmeye ya da sistemde kaynak kullanım oranını maksimize etmeye çalışır, kullanıcıların hizmetten memnuniyetini ölçemez. Sistem-merkezli olup sistem için iyi olan kararlar alırlar. Geleneksel sistem odaklı iş sıralayıcılar, merkezi politikalara dayanır ve genelde tüm sistemin durum bilgisine ihtiyaç duyarlar. Eş zamanlı olarak hem Kaynak sahipleri hem de kullanıcıların objektif fonksiyonlarının gerçekleştirilmesine odaklanmazlar.

Şimdiki iş sıralayıcılar bazı sistem-odaklı parametrelerle iş yükü çalıştırır. Oysa kaynak sahibi kaynağına farklı değer ataması yapabilir ve farklı objektif fonksiyonu olabilir. Benzer olarak kullanıcı da farklı QoS a göre davranmak isteyebilir, pazarlık yapmak isteyebilir. Bu nedenle yazarlar, grid ortamındaki dağıtık kaynak kümelerinin yönetimi için ekonomi tabanlı yöntemi önermişlerdir. Ekonomi modeli, kullanıcı merkezli kaynak tahsisini öne çıkarır.

Önerilen sistemin etkinliğini ölçmek için GridSim aracı kullanılmıştır. GridSim simülasyon aracı ile iş sıralama algoritmalarını değerlendirmek için dağıtık sistem varlıklarının simülasyonu ve modellenmesi mümkündür. GridSim ile modellenen varlıklar listesi yaklaşık aşağıdaki gibidir:

1. Hesaplama düğümleri ve işlemciler
2. Kümeler,
3. İş yükleri,
4. GRIS ve Grid İstatistik servisi
5. Lokal kullanıcılar.
6. GFA, Grid-Federation modeli için RMS sistem yerine.
7. GFA kuyruğu (uzak ve lokal işleri tutar),
8. GFA paylaşılan federasyon dizini

Yazarlar, gerçek iş yükü yürütme izleri ve sentetik iş yükleri ile simülasyonlar yürütmüşlerdir.

4.2.2 Bulut sistemlerinin PBM ile yönetimi

Bulut sistemlerinin politikalarla yönetimi konusunda çalışmalar da artmaktadır. Dağıtık sistemlerin ve telekomünikasyon sistemlerinin yönetiminde kullanılan PBM araçları artık bu platformlara da uygulanmaya başlanmıştır. Politikalar, bulut gibi dinamik ve dağıtılmış ortamlar için oldukça uygundur.

Kullanıcılara, SaaS (Software as a Service), PaaS (Platform as a Service) veya IaaS (Infrastructure as a Service) olarak verilen hizmetler, genellikle kullandıkça öde modeli ile verilir. Geleneksel erişim denetim sistemleri (discretionary, mandatory ve role-based) kullanılsa da dinamik ve esnek sistemler için uygun olmayabilmektedirler. Çoğu sanal kaynak (sanal makina, servis vb.) operatörlerin kontrolü dışında, otomatik olarak oluşmakta veya çoğunlukla kısa sürede ihtiyaç bitince yok olmaktadır. Artık elle idare edilen sistemlerden, politika tabanlı yönetilen otonom altyapılara geçişe ihtiyaç duyulmaktadır. Bu alanda özellikle sanal kaynakların güvenliğinin politikalarla yönetimi konusunda çalışmalar vardır. Artık altyapıdaki donanımsal sorunlar veya işleyişten çok, platformlarda çalıştırılan uygulamalara odaklanılmaktadır.

[134] yazarları, bulut yönetim güvenliği için bir model önerir. Politika tabanlı erişim modeli, bulut yetki alanları (*tenantlar*) arası hızlı yetkilendirme, görev yönetimi vb. destekler sunar. [135] çalışması ise bulut ortamındaki, yöneten/yönetilen varlıklar, olaylar ve politikalar için formal bir model tanımlar. Birden fazla otonom yönetici arasındaki politika pazarlık arayüzleri bile içerir. Basit dil yapısı örneği: *On event: E if (Set of Conditions) then { Ordered Set of Actions }*

Bulut sistemlerinin üzerine kurulduğu fiziksel altyapıların çok geniş, karmaşık olabilir ve yapılandırmaları zahmetli ve hataya eğilimli olabilir. Ayrıca, çok farklı kullanıcıların, bir anda aynı fiziksel kaynağı paylaşıyor olması güvenlik risklerine sebep olabilir. [136] ve [137], bulut yapılarının politika ile yönetilmesi konusundaki gereksinimleri, zorlukları ve çözümleri inceleyip kendi modellerini önermişlerdir. Ticari ürünler dünyasında da bulut yapıları başta olmak üzere PBM ürünleri boy göstermiştir. Bunlardan bir kaçından aşağıda bahsedilmiştir.

Cloudsoft AMP (Application Management Platform), açık kaynak kodlu, PBM destekleyen bir bulut uygulamaları yönetim ürünüdür. Politikalarla, işleri kaynaklara yerleştirme kriterleri, yüksek kullanılabilirlik gereksinimleri, ücretlendirme kısıtları, öncelikler ve servis kalitesi tanımlanabilir.

Politikalar sensörlere kaydolabilir ve kaydoldukları olaylarla tetiklenebilirler. Girdiler, çıktılar ve eşik değerleri özelleştirilebilir.

Bu sistemde kullanılan politika türleri aşağıda listelenmiştir:

- *AutoScaler Policy*: Varlıkların boylarını koşullara göre değiştirir. Örneğin, bir Tomcat Sunucuya gelen birim zamandaki istek sayısı eşik değeri aşınca hesaplama kaynağını artırmak.

- type: org.apache.brooklyn.policy.autoscaling.AutoScalerPolicy

brooklyn.config:

metric: webapp.reqs.perSec.perNode

metricUpperBound: 3

metricLowerBound: 1

resizeUpStabilizationDelay: 2s

resizeDownStabilizationDelay: 1m

maxPoolSize: 3

- *ServiceRestarter*: Çöken yazılımları, belli süreler ve tekrarlarla yeniden başlatmayı tanımlar.
- *StopAfterDuration Policy*: Bir varlığın yaşam süresini sınırlar.
- *CreateUser Policy*: Belli koşullarda bir varlığa yeni kullanıcılar oluşturmak için tanımlanır.
- *AdvertiseWinRMLogin Policy*: Yeni kullanıcı eklendiğinde bir sensör oluşturarak kullanıcının giriş bilgilerini oluşturur.
- *SshMachineFailureDetector* ve *ConnectionFailureDetector*: Bir makinayla bağlantı koptuğunda ilgilileri bilgilendirmek için kullanılır.
- *ServiceReplacer*: Hataya sebep olan varlıkların temizlenmesi için kullanılır.
- *ConditionalSuspend Policy*: Bir politikanın, sensör çıktılarına göre askıya alınması ve tekrar aktifleştirmesi için kullanılır.
- *LoadBalancing Policy*: Çalıştırma ortamlarındaki yük durumlarına göre işlerin, kaynaklar aralarında taşınması koşullarını tanımlamakta kullanılır.

Bir diğer örnek olan Oracle Enterprise Management [138] yazılımı, bulut yönetimi için, Performans ve İş Sıralama politikaları sağlar. Politika örneği: Bir Sanal Makinayı, her Cuma günü saat 18:00'de sonlandır

BMC Cloud Lifecycle Management [139] aracı, bulut kullanımına sunulan bir kaynağın tanımlanması ve bütün yaşam çevrimini yönetir. Bu aracın Governor adlı akıllı PEP bileşeni, işlerin kaynaklara dağıtılması ve yönetilmesini politika tabanlı olarak yerine getirir. Politikalar kaynakların etiketlerini içerir. Kullanıcı özellikleri ve ihtiyaçlarına göre otomatik kaynak seçimi yapılır. *Compute Pool, Network, Network Container, PaaS, Virtual Disk* tahsisleri niyöneten politikalar vardır.

4.2.3 Merkeziyetsiz sistemlerde PBM ile yönetim

Öte yandan literatürde, farklı amaçlar için PBM ve Blokzincir teknolojilerini bir araya getiren çalışmalar bulunmaktadır. Yakın zamanda yayınlanan birçok çalışma [140-152] blokzinciri, bir politika deposu veya sadece yönetim için sınırlı yeteneklere sahip bir PEP olarak kullanırken, [140, 142-147] çalışmaları ise blokzinciri, IoT cihazlarının erişim kontrol politikalarını yürütmek için de kullanır. Ayrıca, hizmetler ve kişisel verilere erişim kontrolünde blokzincir ve PBM teknolojisini kullanmaya odaklanan çalışmalar da [141, 148, 149] bulunmaktadır. Bu çalışmalarda blokzincir teknolojisinin kullanım amacı değişkenlik göstermektedir:

- Politika Uygulama Kanıtlarının Depolanması:[153], veri gizliliği politikalarını uygulayan bir erişim kontrolü mekanizması önermektedir. Bir Hyperledger Fabric blokzinciri, yalnızca politika uygulamasının kanıtı için depolama ortamı olarak çalışır.
- Politika Depolama: [143] ve [145], UTXO [153] blokzincirinde politikalar yayınlar ve bunları zincir dışı PEP'ler aracılığıyla uygular. [145], politika erişim haklarını token olarak modeller. Token sahipleri onu başka bir aktöre teslim edebilir veya kaynağa erişmek için harcayabilir. [151], akıllı şehir uygulamaları için XACML kurallarını depolamak için Hyperledger Fabric blokzincir platformu kullanır.
- Politika Depolama ve Uygulama: [143], federe hizmet altyapılarında ağ ve güvenlik yönetimi için Fabric blokzinciri kullanır. Hizmet sağlayıcılar, politikaları, kullanıcılar için hizmetleri başlatmak/durdurmak için harici kaynaklara talimat veren PEP rolündeki akıllı sözleşmeler olarak kullanır. Burada amaç, erişimde şeffaflık sağlamaktır. [146], her bir IoT kaynağı için Ethereum akıllı sözleşmesi olarak gerçekleşmiş PEP'ler kullanan bir erişim kontrol çerçevesi sunar. Akıllı sözleşmelerde depolanan ve uygulanan çok

basit politikalar kullanır. [147] ve [148], politikaları, özelleştirilmiş bir blokzincirin bloklarında depolar. [150], bir Ethereum akıllı sözleşmesinde erişim kontrol politikalarını depolayan bir prototip sağlar.

- Politika Depolama ve Karar Verme: [152], Hyperledger Fabric blokzinciri üzerinde, bir Nitelik Tabanlı Erişim Kontrolü (ABAC) sistemi önermektedir. Politika kararı ve veri havuzu bileşenleri, zincir dışı PEP'lerle etkileşime giren akıllı sözleşmelerdir.

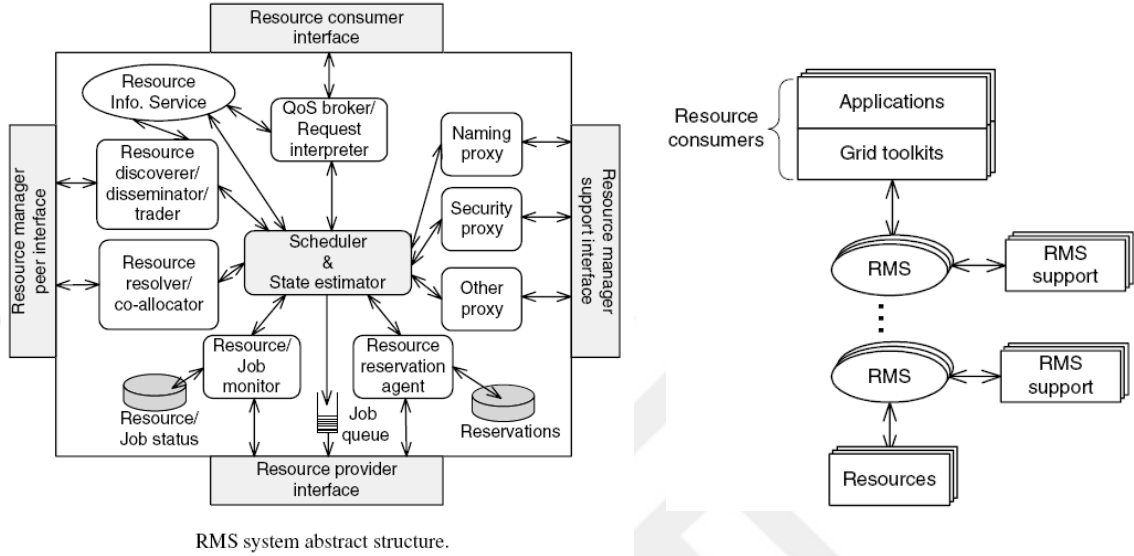
Yukarıdaki çalışmaların çoğu (XACML [154] gibi karmaşık dilleri kullanan birkaçı dışında) ilkel politika dilleri kullanır ve Hemen hepsi erişim denetimini amaçlar.

4.3 Evrensel YBH Modeli

YBH sistemlerinin politika tabanlı yönetimi konusunda araştırmalar yapmak için, bu sistemleri farklı koşullar altında testlere tabi tutmak ve ölçümlemek gerekir. Bu iş için bilinen YBH platformlarının yazılımlarının değiştirilerek PBM ile yönetilebilir hale getirilmesi ilk akla gelen seçenektir. Ancak tez süresi içine sığmayacak uzunlukta bir çalışmadır. Hem de bu değerlendirmeler için tekrarlı, kontrollü deneyleri gerçek YBH sistemleri üzerinde yapmak her zaman mümkün olmayabilir. Çünkü sistem bileşenlerinin kullanılabilirlik durumları sürekli değişir. Kullanıcıların eylemlerini kontrol altında tutmak neredeyse imkansızdır. Dahası, bu sistemlerin birim zaman maliyetleri çok yüksek olduğu için, akademik çalışma deneyleri için fırsat bulmak genelde mümkün olmaz. Bu nedenle bu sistemler üzerindeki çalışmalar, genellikle bu sistemlerin geçmiş işlem kayıtları üzerinden geriye doğru analizlerle yapılır. Bu sebeplerle, tez çalışmasında, gerçek YBH sistemleri yerine, bunların simülasyonları ile çalışma yolu seçilmiştir. Bu seçim ile ilerleyebilmek için YBH simülasyonlarını mümkün kılan simülatör yazılımı da geliştirilmiştir. Bu bölümde, geliştirdiğimiz simülatöre dayanak oluşturan YBH kavramsal modeli tanıtılmıştır.

Literatürde, YBH mimarileri için bazı kavramsal model önerileri vardır. Hem Küme hem de Gridleri tanımlamada kullanılabilecek bir evrensel model [18] tarafından Şekil 4.20'deki gibi verilmiştir. Bu soyut model, çeşitli fonksiyonel birimler ve dört adet arayüze (RCI, RPI, RMSI, RMPI) sahiptir. Bu modeldeki kaynak (resource), bir RMS kontrolünde çalışan bilgisayar, şebeke veya ağ servisi olabilir. Bu modelin yapı taşı RMS'lerdir. Kaynak tüketicisi ise RCI arayüzü üzerinden RMS ile etkileşen daha üst

düzey başka bir RMS veya bir YBH istemci uygulaması olabilir. Kaynak sağlayıcısı ise RPI arayüzü üzerinden RMS ile etkileşen gerçek bir kaynak veya hiyerarşik olarak daha aşağıdaki başka bir RMS olabilir. İsim servisi ve güvenlik gibi destek fonksiyonlarına RMSI arayüzü üzerinden erişilir. RMPI arayüzü ise kaynak keşfi, kaynak kullanım ticareti ve ortak kaynak kullanımı gibi çeşitli protokolleri içindir.

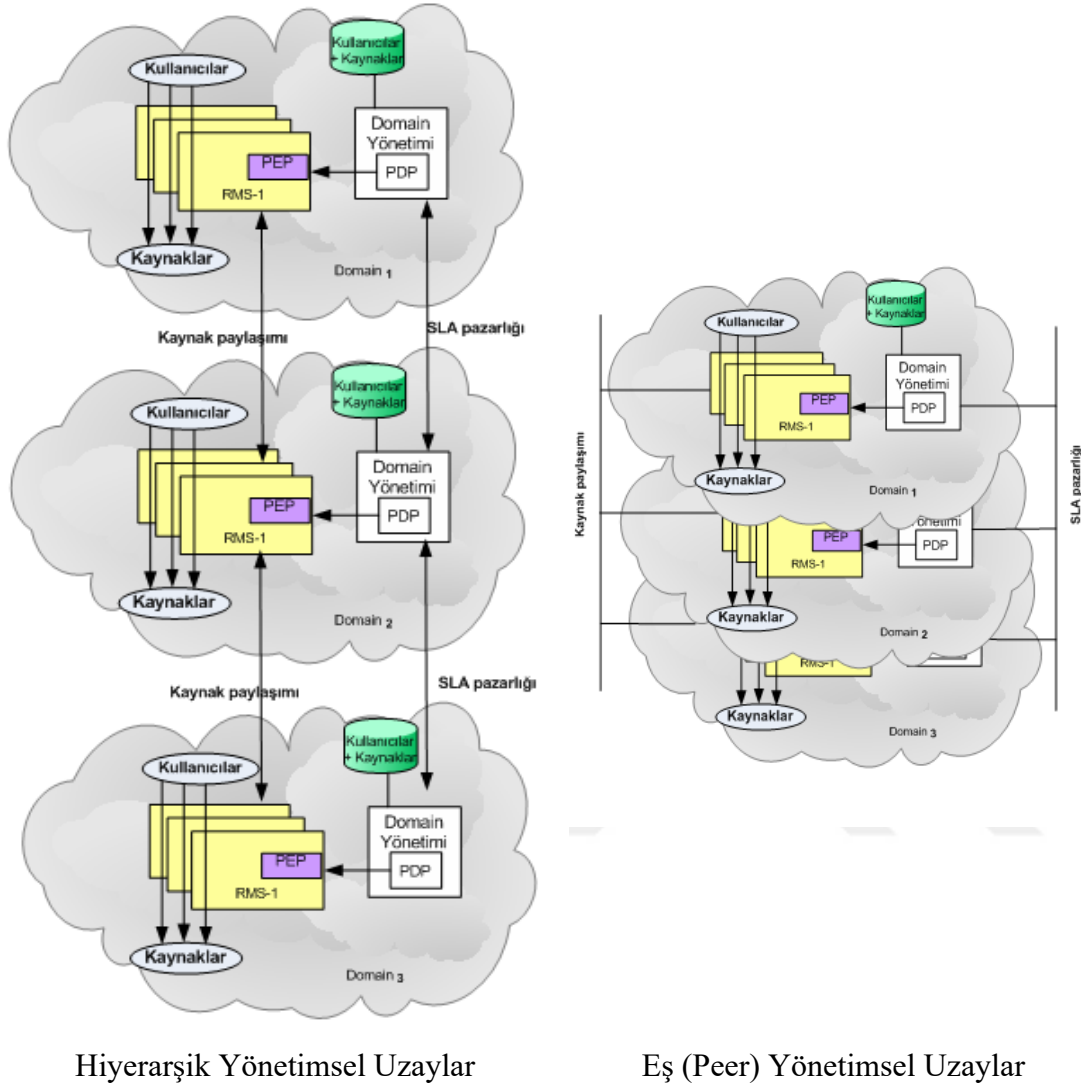


Şekil 4.20 : RMS’ler için önerilen soyut model ve gridlerde kullanımı [18].

Bu soyut model kullanılarak, bir veya daha fazla hiyerarşide RMS’ten oluşan çeşitli YBH mimarileri modellenebilir. Ancak YBH sistemlerinin yönetimiyle ilgilenmediği için, bu modelde farklı kişi veya kurumlara ait yönetim uzayları (administrative domains) kavramı ihmal edilmiştir. Modele, farklı yönetim uzaylarındaki kaynakları modellemeye ek başka eklemeler yapılması da gerekmektedir. Farklı organizasyonların kullanıcıları modellenebilmeli, işlerin kaynaklara erişim yetkileri hesaba katılmalıdır [155]. Ayrıca, işleri etkileyebilecek olan, YBH sistem bileşenleri arasındaki semantik kısıtlar da modellenebilmelidir.

Farklı sahiplere ait YBH uzaylarının, kendi yöneticileri ve yönetim mekanizmaları vardır. Eğer sistemler politikalarla yönetiliyorsa, kendilerine özel PBM altyapıları da vardır. Her uzay, kendi politikalarına göre yönetilir. Şekil 4.21’de de gösterildiği gibi, bu etki alanları farklı mimarilerde bir araya gelerek kaynaklarını birbirleriyle paylaşabilirler. Yönetim uzayları, bir dizi RMS’in kontrolündeki kaynakların değişik topolojilere göre bağlanmasından oluşabilir. Kaynakların etki alanları arasında paylaşımı, etki uzaylarının yönetimleri arasında gerçekleşen pazarlıklar ve üzerinde anlaşılan SLA’lar vasıtasıyla olur. Yönetim uzayları arası topolojik alternatifler

yanında, alan içinde yer alan RMS'ler arasında da çeşitli topolojik senaryolar mümkün olabilir.

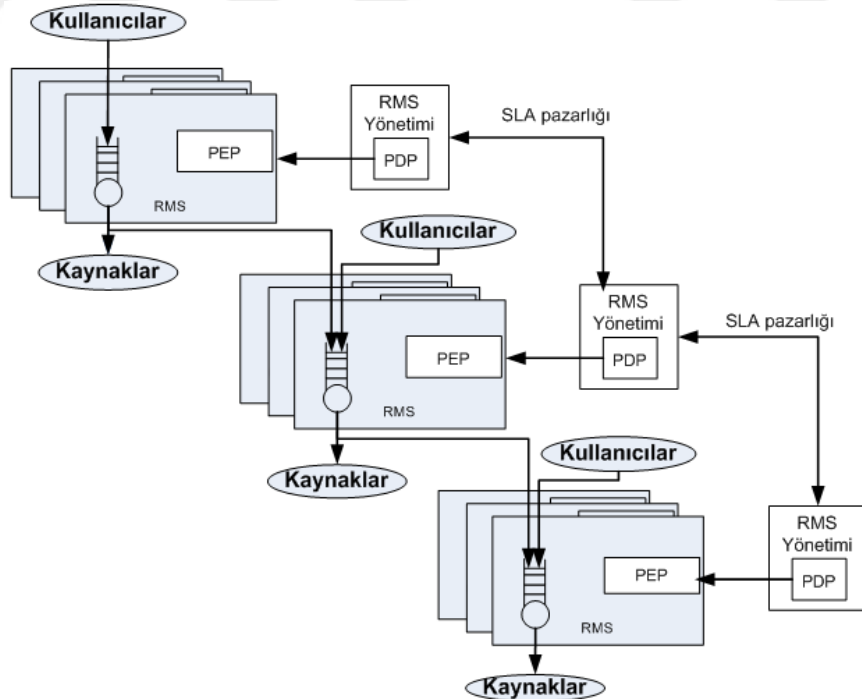
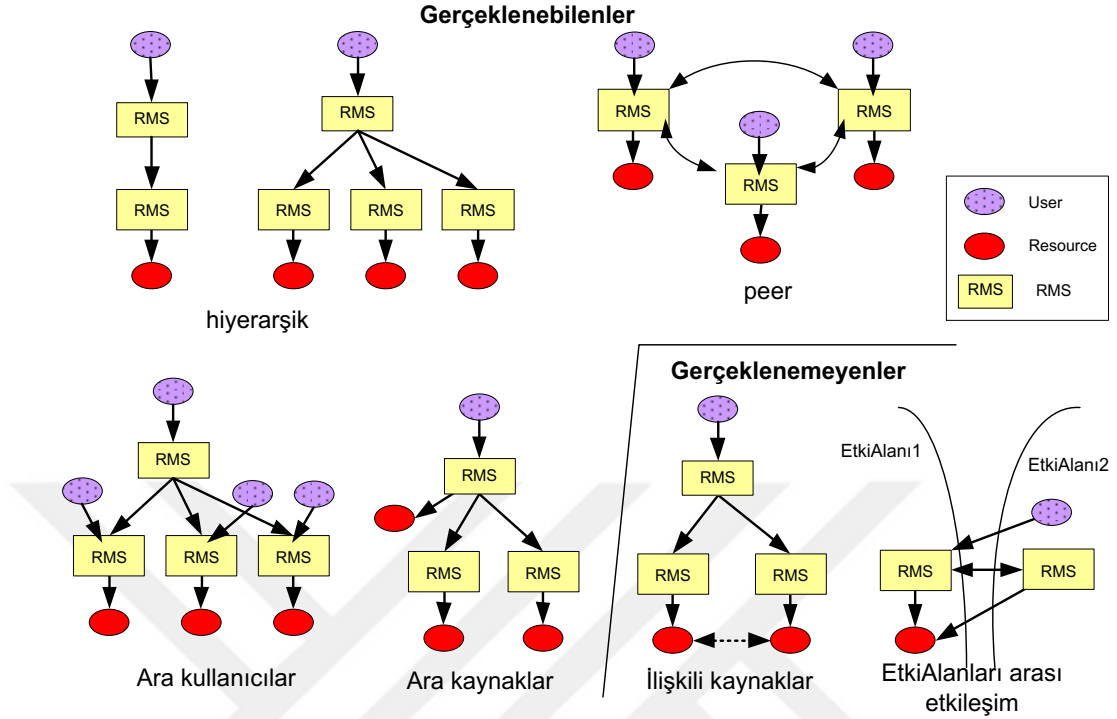


Şekil 4.21 : Yönetimsel (Administrative) uzaylar arası bağlaşımlar.

[18] modelinde, RMS'ler arası topolojik senaryolara izin vardır. Bu modelde gerçekleştirilemeyen senaryolar Şekil 4.22'de özetlenmiştir. Bu modelin ihmal ettiği yerel kullanıcılar ve yerel kaynaklar da dikkate alındığında Şekil 4.23'deki gibi bir mimariye ve PBM senaryosuna ulaşılır.

Tasarlanacak genel amaçlı mimari ile hem merkezi, hem dağıtık hem de hiyerarşik YBH sistemleri oluşturulabilir ve yönetilebilir olmalıdır. Sadece *Scheduler* değil, YBH'nin tüm bileşenleri yönetilebilmelidir. Mesela iş sıralama politikalarında, Erişim Denetimi de hesaba katılmalıdır. Aksi halde iş sıralayıcının, kullanıcının yetkisi

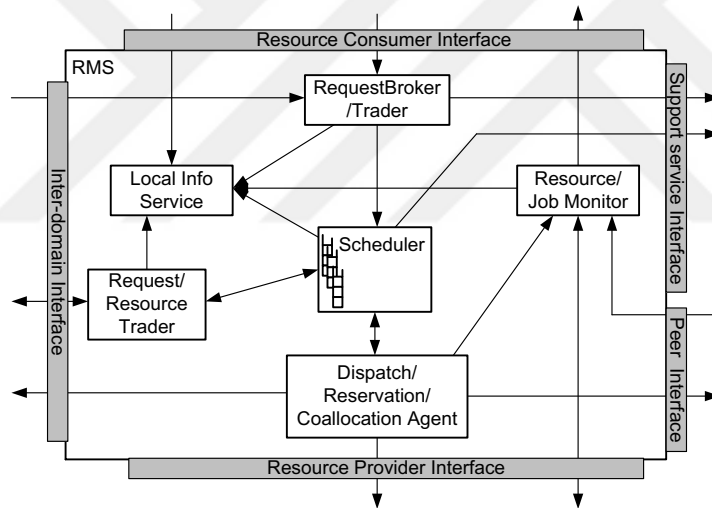
olmayan bir kaynağa atadığı iş ancak kaynağa gönderildikten sonra anlaşılabilir. Farklı konulardaki politikalar birlikte değerlendirilmeli ve uygulanmalıdır.



Yukarıdaki bütün hususlar dikkate alınarak, [18]'de önerilen modeli baz alan, politika tabanlı olarak yönetilebilir genel geçer bir YBH sistem soyutlama modeli

geliştirilmiştir. Bu modelde aşağıdaki basitleştirme ve geliştirmeler yapılarak daha fazla YBH senaryosunun modellenilebildiği Şekil 4.24'deki yeni model elde edilmiştir:

- Bu model ile merkezi, hiyerarşik ve dağıtık topolojilerin yanı sıra etki alanları arası iletişim de dikkate alındı. Yönetim Uzayları arası kaynak kullanım senaryolarını modelleyebilmek için farklı yönetsel uzaylardaki RMS'lerin çeşitli protokollerle etkileşimine izin veren *Inter-Domain Arayüzü* eklendi.
- RMS'in bütün bileşenlerinin politika tabanlı olarak yönetilebilmesini sağlamak üzere PEP Arayüzü adlı yeni bir Destek Servisi eklendi. Böylece, kendi RMS modelimizin politika uygulama ile bağlantısı sağlanmış oldu.
- Eş (peer) RMS'deki kaynakları gözleyebilmek için gerekli yetenek, Resource/Job Monitor'e eklendi.
- Kolay gerçekleştirilebilirlik için, fonksiyonellik olarak birbirine yakın bazı modüller birleştirilerek model sadeleştirildi.



Şekil 4.24 : Soyut RMS modeli ([18] modelinin genişletilmiş sürümü).

Bu RMS'lerimizi hiyerarşik veya eş (peer) olacak şekilde birbirlerine bağlayabiliriz ve/veya farklı yönetim uzayları içinde olacak şekilde gruplayabiliriz. Üç RMS içeren bir senaryo örneği Şekil 4.25'de gösterilmiştir.

Elde edilen yeni RMS yapısında, ortaya çıkan evrensel YBH modeli varlıklarına ilişkin *Bilgi Modeli*, Şekil 4.26'da gösterilmiştir. Bu modeli simüle etmek için, detayları Bölüm 5.3'de açıklanan bir simülasyon aracı geliştirilmiştir.

Simülasyonun doğruluğu, modelleme sonuçlarının, ilgili gerçek sistemin saha sonuçları ile karşılaştırılması ile anlaşılabilir. Bu amaçla çeşitli tanınmış YBH

The figure illustrates the architectural models of the Resource Management System (RMS). It shows three detailed RMS architectures (RMS₁, RMS₂, and RMS₃) and a Common Services block.

RMS Architecture Components:

- User:** Interacts with the Resource Consumer Interface.
- Resource Consumer Interface:** Connects the User to the RequestBroker/Trader.
- RequestBroker/Trader:** Manages requests and trades.
- Local Info Service:** Provides local information.
- Scheduler:** Schedules resources.
- Dispatch/Reservation/Coallocation Agent:** Manages resource allocation.
- Resource/Job Monitor:** Monitors resource usage and job progress.
- Resource Provider Interface:** Connects the RMS to the Resource.
- Resource:** The physical resource being managed.

Interfaces:

- Support service Interface:** Connects RMSs for support services.
- Peer Interface:** Connects RMSs for peer-to-peer interactions.
- Inter-domain Interface:** Connects RMSs across different domains.

Common Services:

- Security
- PBM (Policy-Based Management)
- Statistics

Summary Diagram:

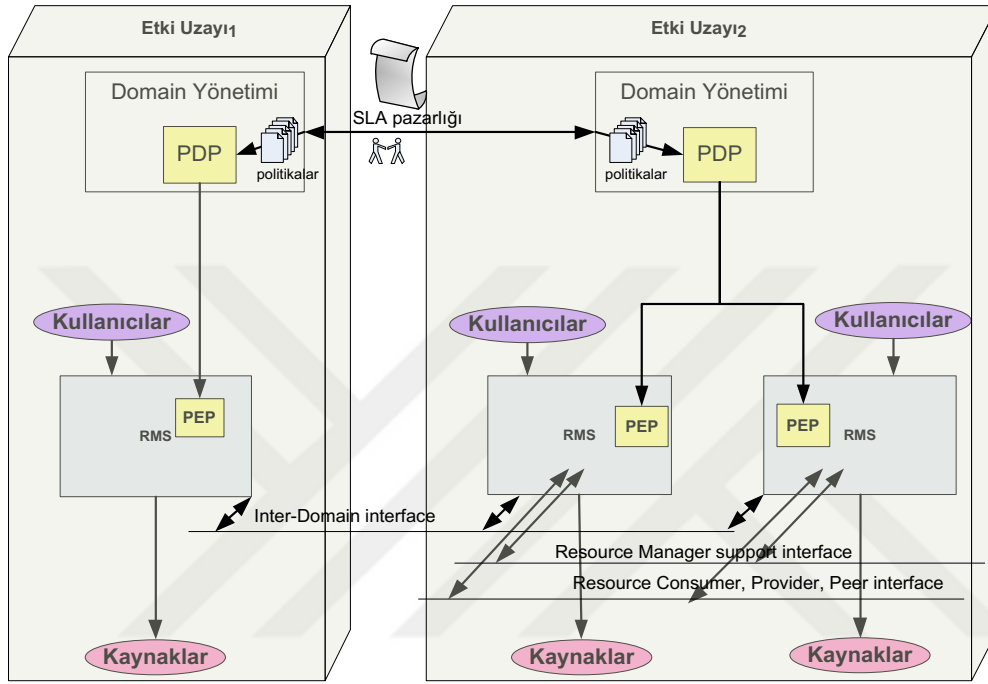
- Hierarchical:** User → RMS-1 → RMS-2.
- Distributed:** User → RMS-1 and RMS-3.

The diagram illustrates the SRM architecture with the following components and relationships:

- «SimulationEntity» User**: Associated with **RMS** via a **users** relationship (multiplicity 1 to *).
- AdministrationDomain**: Contains attributes `-virtualOrganization` and `-domainName`. Associated with **RMS** via an unnamed relationship (multiplicity 1 to *).
- PDP**: Associated with **PEP** (multiplicity 1 to *) and **RMS** (multiplicity 1 to *).
- CommonService**: Generalized by **«SimulationEntity» InformationService** and **«SimulationEntity» Security**.
- «SimulationEntity» PEPAdapterEntity**: Associated with **RMS** via an **adapterEntity** relationship (multiplicity 1 to 0..1).
- Queue**: Associated with **RMS** via a **queues** relationship (multiplicity 1 to 1..*).
- RMS** (SimulationResourceManager):
 - Has a **source** association to **«SimulationEntity» source** (multiplicity 1 to *).
 - Has a **peer** association to **«SimulationEntity» Broker** (multiplicity * to *).
 - Has a **parent** association to **«SimulationEntity» Scheduler** (multiplicity * to 0..1).
 - Has a **dispatcher** association to **«SimulationEntity» Dispatcher** (multiplicity * to *).
 - Has a **resourceTrader** association to **«SimulationEntity» ResourceTrader** (multiplicity * to *).
 - Has a **resourceMonitor** association to **«SimulationEntity» ResourceJobMonitor** (multiplicity * to *).
 - Has a **localInfoService** association to **«SimulationEntity» LocalInformationService** (multiplicity * to *).

93

Bu modelin PBM ile yönetimi için PBM bileşenlerinin model elemanları ile arabağlaşımları Şekil 4.27’de gösterildiği gibi yapılabilir. Bu yönetim modeli sayesinde, hem [118] çalışmasındaki PEP senaryolarını hem de genel amaçlı bir YBH yönetimi senaryosunu gerçekleyebiliriz. [118] çalışmasında hiyerarşik PEP’ler ve bunlar arasında farklı dil/formatta tanımlanmış politikaların transferi/dönüşümü yapılmaktadır. [118] çalışmasındaki mimari, bizim modelimizde farklı iki otonom



Şekil 4.27 : Evrensel YBH modelinin PBM yönetimi (örnek senaryo).

uzay olarak gerçekleştirilebilir. Biz her yönetim uzayı için tek bir PDP olduğunu varsayıyoruz. Her yönetim sunucusu, kendi otonom yönetim uzayı içinde geçerli olacak politikaları tanımlar. Yönetim uzayları arası iş gönderme ve kaynak paylaşımı, politikalar kontrolünde yapılır. Yalnız Kullanım Politikaları (Resource Usage Policy) ile kaynaklara erişim kotalarını değil tüm sistem bileşenlerini PBM ile yönetmek mümkün olur. İş yüklerinin, hangi etki alanında olduklarından bağımsız olarak çalışabilmesine de olanak sağlayacak bir yönetim sistemi hedeflenmiştir. Evrensel modele göre, sistemde yönetilmesi gereken varlıklar aşağıdakileri içerir:

- İş yükü (sıralama, başlatma, durdurma, silme, taşıma, önceliklendirme)
- İş yükü kuyrukları (strateji, öncelik mekanizması vb.)
- Rezervasyonlar
- İş Sıralayıcı
- QoS/SLA Aracıları

- Kaynak tahsisleri ve kısıtları
- Erişim denetimi

Bunların yanında Kullanıcılar ve kaynakların da yönetilebilmesi, sistemin bütünleşik yönetilmesi açısından önemlidir. Bu modelin sahada kullanılacak olgunluğa gelmesi için aşağıdaki listedeki konuların halledilmesi gerekir:

- Hiyerarşi derinleştikçe artan, sistem kaynaklarını ve kullanıcı etkinliklerini izleme (monitoring) yükü oluşur. Etkin bir izleme mekanizması kurulmalıdır. Güncel kaynak durumu bilgisini her RMS'nin bilmesi zorlaşır. Kaynakların global bir görüntüsünü oluşturmak zorlaşır. Gözleme konusundaki bu problemler çözülmelidir.
- Otonom uzayları arası SLA pazarlıkları ve bu pazarlık sonuçlarının otomatik olarak politikalar şekline dönüştürülmesi için mekanizmalar gerekir.
- Yönetim uzayları arası politikaların, yerel politikalarla çakışmasının önlenmesi/çözülmesi gerekir.

4.4 Police Modeli ile YBH Yönetimi

Önceki bölümde tanıtılan evrensel YBH modelin simülasyonlarının politika tabanlı yönetim bileşeni olarak kullanmak için, bu tezin hazırlığı kapsamında geliştirilen Police PBM aracı tercih edilmiştir.

Tez çalışmasında, YBH politikalarını Police yazılımı [7, 8] ile tanımlamak ve tanımlanan bu politikalarla sağlanan yönetimsel etkileri, YBH simülasyonları üzerinde göstermek amaçlanmıştır. Simülasyon geliştirme çalışması kapsamında yürütülen literatür araştırmalarından, var olan simülasyon araçlarının, yine literatür araştırmalarından derlenen YBH özelliklerinin tümünü bir arada destekleyemediği görülmüştür. Bu nedenle, evrensel YBH mimarilerini de simüle edebileceğimiz kendi simülasyon aracımız geliştirilmiştir. Ancak geliştirdiğimiz bu evrensel simülasyon altyapısı ile modellediğimiz YBH senaryolarını, Police PBM aracı yönetebilmek için aşılması gereken bir konu daha vardır. Police yazılımının kendisi, simülasyon varlığı olamayacak bir gerçek dünya varlığıdır. Dolayısıyla simülasyon bileşenleri (YBH senaryoları) ile gerçek dünya yazılımı (Police modeli) arasında etkileşimin sağlanması gerekmektedir. Simülasyon dünyasında var olan araçlar ve modeller, gerçek dünya yazılımları ile simülasyon uzayında çalışan yazılımların, bir simülasyon oturumu

kapsamında etkileşmesi için formal bir çerçeve sunmamaktadır. Bu nedenle, geliştirdiğimiz simülasyon ortamının bu özelliğinin olması gerekiyordu. Geliştirdiğimiz, HeteroSim [9] adlı simülasyon platformu, ayırık zamanda, hem gerçek zaman (*real-time* değil *real clock*) yazılımlarını hem de simülasyon varlıklarını içeren oturumlar düzenlemeye olanak sağlar. HeteroSim'in bu özelliği sayesinde, çeşitli YBH senaryolarında, simülasyon varlığı olarak kodladığımız nesneleri, Police'in politika uygulama elemanları yardımıyla yönetme olanağı elde edilmiştir. Bunu sağlayabilmek için Police modelinde değişiklikler yapılmıştır. Böylece, Police modelinin, YBH yönetimi konusundaki etkinliğini görebileceğimiz simülasyonlar düzenleyebilme mümkün olmuştur.

Öte yandan, simülasyonlarda kullanacağımız YBH yönetim politikalarını belirlemek için literatür taraması yapılarak çalışılacak politika türleri belirlenmiştir.

HeteroSim ve Police modelinin entegrasyon detayları, Bölüm 5.3.4 'de verilmiştir. Simülasyonlardaki iş yükü izleri (trace) olarak benzer sistemlerde kullanılan iş yüklerinin kullanılmasına özen gösterilmiştir. Benzer sistemlerle daha önce yapılmış yürütmeler, aynı iş yükü ve benzer koşullar altında bizim prototip sistemle de tekrarlanmıştır. Simülasyonlar çalıştırılıp belirlenmiş metrikler ölçülerek politika tabanlı yönetimin etkinliği gösterilmiştir.

Police altyapısında, tez kapsamında aşağıdaki değişiklikler yapılmıştır (Değişikliklerden bazıları YBH yönetimine özel olmayan genel iyileştirmelerdir):

- Politikalara hangi SLA ile ilişkili olduklarını belirten tanımlayıcı alan (ilişkili-SLAId) eklendi.
- E-Code'lar içinde kullanmak için yeni arayüz sınıflar tanımlandı. Mesela, Schedule, srcResource, dstResource, requiredQoS vb. Genel amaçlı, tüm sistemlerdeki tanımlarını kapsayacak şekilde, HPC, JOB, QUEUE, SCHEDULER, QOS, vb. arayüz sınıfları hazırlandı.
- YBH uzayı bileşenleri için E-Code şablonları hazırlandı.
- Politika kısıt bölümünde, YBH ile ilgili her türlü fonksiyonun kısıt ifadesi içinde kullanılabilmesi sağlandı. Bunu sağlamak için sistemi gözleyen mekanizmanın esnek ve genişletilebilir olması gerekir. Toplam kaynak boş kalma süresi, kaynaklarının iş yükü durumu, işlerin ortalama boyu gibi

istatistiksel fonksiyonları, politikaların koşul (When) kısmında kullanabilmek için istatistiksel operatörler ve fonksiyonlar eklendi.

- Aktörsüz (Impersonal) politikalar kullanılarak sistemde otomatik olarak yapılması gereken eylemlerin gerçekleşmesine olanak sağladı. Örneğin, iş sıralayıcının birden fazla çalışma modu (strateji) arasında, yürütme anındaki koşullara göre hangisine geçileceğine karar vermek ve performansı artırmak için aktörsüz politika örneği: *Scheduler.setStrategy("GREEDY") when hpc.awt() > 10.*

4.4.1 YBH politika türleri

Bölüm 4.2’de incelenen literatür çalışmalarından, bir YBH sisteminde kullanılan politika sınıfları derlenerek bu bölümde listelenmiştir. Bu çalışmalardaki benzer tipler elenmiş veya birleştirilmiştir. Prototip gerçeklemlerimizde bu politikaların mümkün olduğunca fazlasının desteklenmesi hedeflenmiştir.

- Oturum (Session) politikaları: Her bir kaynak kullanım oturumu için, eylemleri belirler. Kaynak ve sistem sahibi tarafından tanımlanırlar.
- Kullanım (Usage) politikaları (CPU, FileSystem) [119]
- Muhasebe ve Fiyat politikaları
- Dosya yönetim politikaları: (File caching, cache replacement, replication) işlerle ilişkili dosyalarla ilişkili eylemleri tanımlar.
- Servis politikaları: Bir sonraki işi seçmek için kullanılan lokal kısıtlar.
- İş sıralama (Scheduling) politikaları
- Kaynak seçim politikaları
- Hesap (Account) politikaları: Kullanıcıların kaynaklara erişim denetimi politikalarıdır.
- Uygulama (Application) politikaları: Çalıştırılan uygulamalarla ilgili kısıtları tanımlayan politikalardır. Örnek: Bir işin kaç alt parçaya bölüneceğini tanımlamak.
- QoS politikaları: Kaynağın sahibi ve kullanıcısı açılarından hizmet metrikleri uygulama amaçlı politikalardır. Sistem, SLA’da tanımlanan QoS seviyelerini garanti etmeye çalışır. YBH sistemleri genellikle ya cevap süresini minimize etmeye ya da sistemde kaynak kullanım oranını maksimize etmeye çalışır. Sistem-merkezli iş yükü planlayıcılar, sistem için iyi olan kararları alırlar. Genelde merkezi politikalara işleyen bu sistemlerde kullanıcıların, QoS parametreleri

belirtmesine izin verilmez. Kullanıcıların gönderdikleri işler saatlerce kuyruklarda bekleyebilir. Bunun aksine, mesela [133]'de tanıtılan ekonomi tabanlı iş sıralama mekanizmasında, kaynaklara arz/talep durumuna göre dinamik fiyatlar biçilir (kullanıcı merkezli bir model). Bu yöntemde, hem fiyat hem de performans kriterleri SLA tanımlarında kullanılabilir.

4.4.2 Politikaların kısıt ifadeleri

Literatür araştırmasında tesbit edilen kısıt ifadeleri aşağıda benzerleri bir araya gruplanarak paylaşılmıştır.

1) YBH politikalarında çeşitli kısıt ifadeleri bulunabilir. Örnekler:

- Her iş yükü, kaynak tipleri üzerinde gereksinimler belirtebilir (Örnek: A işi, hızı 1600 MHz'den büyük olan CPU'ya atanmalı).
- İş yükü, yürütme zamanı ve yeri konusunda kısıt belirtebilir.
- İş yükü, şartları sağlayan kaynaklar arasında tercih de bildirebilir.
- İş yükü, bağlantılı kaynakların dinamik bazı özellikleri için de kısıt tanımlayabilir.
- Data Gridlerde verilerin bulunduğu konum da bir iş yükü kısıtı olabilir. Veri ile ilgili kısıt veya politikalar tanımlanabilir.
- İş yükleri arasında da bağımlılıklar olabilir. İş yüklerinin yürütme sırasını belirtmek için de önce gelme kısıtı tanımlanır (precedence constraints).

2) YBH kaynaklarının da çalıştırılacak iş yükü tipleri, zaman tercihleri, sistem topolojisi ile ilgili tercih kısıtları olabilir.

3) YBH sistemlerinde kullanılan ağ altyapısının yönetimi, gridler için kullanılan yöntem ve araçlarla yapılmaz. Şu anki YBH politika dilleri, şebekenin kendisi ile ilgili politika tanımlamaya olanak sağlamaz. Oysa kaynaklara erişim ağ üzerinden gerçekleştirildiği için, YBH operasyonlarında, ağ alt yapısının da yönetilebilmesi gerekir. [107]'deki gibi önerilerde, grid politika dili, şebekeye ilişkin QoS yönetimine de dolaylı bir destek sunar. Grid politikaları, şebeke politikalarına da dönüştürülür. Bizim modelimizde, bu kuralları tanımlamak yerine E-Code yöntemi kullanılabilir (Bkz. Bölüm 3). Diğer yöntem ise tüm bileşenleri (şebekeyi, kaynakları, işleri, vb.) yöneten ortak bir PBM ile her katmana (grid, ağ, vb.) ilişkin politika tanımlamaktır.

Literatürde, yukarıdaki türden (kaynaklar ve şebeke ilişkileri vb.) uygulama uzayına ilişkin semantik kısıtları kotarabilmek için, gerçek politikaların yanında, meta-politika adı verilen, *politikalar hakkında politikaların* tanımlanması benimsenmiştir. Bizim modelimiz (Police PBM), sistem yöneticisinin bütün bu kısıtları önceden bilmesini gerektiren ve statik olduğu için bazı durumlarda işe yaramayan bu meta-politika yaklaşımına alternatif olarak, E-Code mekanizması içerir. Police modelinde semantik kısıtlar, E-Code adı verilen program parçacıkları içerisinde kotarılır. Dolayısıyla, kaynaklar ve iş yükleri hakkında tanımlı olabilecek, Uygulama uzayına özel kısıtların (semantic constraints) kotarılması, Police tarafından sağlanır. Sistem yöneticisinin her zaman bilemeyebileceği bu türden semantik kısıtlara aşağıdaki örnekler verilebilir:

- A tipindeki iş yükü, X kaynağına erişmesin ama koşullar K ise erişebilsin.
- 300 cpu gerektirirken, 300 cpu içeren tek parça bir kaynak yok ise bu işi ancak birden fazla kaynak ile yürütebiliriz. Dolayısıyla işin yürütülmesi sırasında, işin doğasını, diğer işlerle ve kaynaklarla olan kısıtlarını ve kaynakların aralarında olabilecek kısıtları dikkate almak gerekir.
- Sistemdeki kaynakların, fazla kaynak tahsisine (*Resource over-provisioning*) uyumlu olup olmadıkları kısıtı. Örneğin 40% CPU payının VO_0 ve 80% CPU payının VO_1 grubuna tahsis edilmesine (%120), kaynakların uyumlu olup olmadığı, kaynakların kısıtı olarak tanımlanabilir.
- Kaynakları, sadece işlerin işlemci ihtiyaçlarına göre tahsis etmek kötü performansa neden olabilir. Mesela veri yöreselliği (data locality) gibi bazı uygulama uzayına özel bilgileri de hesaba katmak gerekebilir. Kaynaklar işlere tahsis edilirken, kaynak bağımlılıkları da değerlendirmeye alınmalıdır.
- Veri Yoğun (*data intensive*) uygulamalar için iş sıralaması yaparken işlerin veriye erişim doğasını da dikkate almak faydalı olur [121]. İşlerin, iş sıralayıcının bilemeyeceği bazı kısıtları olabilir. Örneğin aynı dosyaya, yakın gelecekte ulaşacak bir iş yükü varsa ona öncelik vermek gerekebilir. Sadece veri dosyası erişimi değil, uygulama uzayına özel başka kısıtlar olabilir. Örneğin:

- İşlerin veri dosyası kullanım rejimi dikkate alınsın
- İşlerin paralellik çeşitleri dikkate alınsın
- X dosyası/dizini ancak bazı tip işler tarafından erişilebilir olsun

- X dosyasına/dizinine ancak yazma korumalı erişilebilsin,
- (YBH performansı > X) ise işlerin dosya isteklerine, olduğu anda cevap ver, <X olunca ALG algoritması ile dosya seçimini hızlandır.

4) YBH sisteminin kendisine özel kuralları/kısıtları da bulunabilir. Örneğin,

- X bilgisayarın en fazla şu büyüklükte iş dağıtılabilir,
- Y bilgisayarına iş atandıysa T süre boyunca aynı bölümdeki Z bilgisayarlarına iş atanmaması gerekiyor olabilir.

5) E-Code ile YBH sistemlerinde başka ihtiyaçlara da cevap verilebilir. Örneğin;

- [109]'de olduğu gibi, istekçinin isterlerine (kaynak yeteneği, kaynak adı, servis sınıfı, QoS) uyan kaynakları istekçiye tahsis ederken, “Kaynak İnşaa” kısıtları (hiyerarşik olarak) dikkate alınır (construction policy). Bu kısıtlar:
 - Sistemde tanımlı olan kaynak tipleri içine gömülebilir,
 - Kaynak havuzunun operatörleri tarafından belirlenebilir veya
 - İsteğin bir parçası olarak kullanıcı tarafından belirtilebilir.

Kaynak İnşaa politikaları E-Code olarak gerçekleşirse bir PBM uzayı ile entegre edilme olanağı da elde edilmiş olur. Aksi halde, [109]'deki kaynak inşaa politikaları, diğer PBM sistemleri ile entegre olamayan, kapalı bir sistem olarak kalır. Kaynakların dinamik özelliklerinin değerleri ancak yürütme anında bilinebilir ki bu bilgiler ancak gerçek bir PBM sisteminde erişilip kullanılabilir. [109]'da kaynak inşaa politikalarını uygulamak amacıyla önerilen sistemin bu eksiği, gerçek PBM aracı ile giderilebilir.

İş yüklerini ayrı ayrı planlayan on-line iş sıralayıcılarda da E-Code kullanılabilir. *Match-making* tarzı kaynak yönetimi, her gelen iş yükü için ilk boş kaynağı bulur. Ancak bu parçalı atama, kuyrukta bekleyen bir sonraki işin kaynak bulamamasına neden olabilir.

5. HETEROSİM HETEROJEN SİMÜLASYON MODELİ

5.1 Giriş

YBH sistemleri, genellikle pahalı ve üzerlerinde akademik deneyler yapmanın genellikle mümkün olmadığı sistemlerdir. Bu amaçla pek çok simülasyon aracı geliştirilmiştir. Geliştirilen araçların çoğu, tek tip bir YBH sistemine veya onun bir alt bileşenine özeldir. Bütün YBH modellerini destekleyen evrensel bir araç yoktur. Bu eksikliği gidermek için bu tez çalışmasında geliştirdiğimiz evrensel simülasyon modeli sayesinde, bilinen YBH sistemlerinin simülasyonunu yapabilmeye olanak sağlanmıştır. Bu evrensel modelin detayları Bölüm 4.3’de, geliştirilen simülasyon ortamının tanıtımı ise Bölüm 5.3’de verilmiştir.

Ölçeklenebilirlik ve Doğruluk, simülasyon araçları için çok önemli iki kavramdır [15]. Her ikisinin de aynı anda yüksek olması zordur. Simülatörlerin ölçeklenebilirliğini tesbit etmek, doğruluğunu tesbit etmekten daha kolaydır. Kullanılan modelin doğruluğunu artırmak ve simülasyonda daha gerçekçi veriler kullanmak simülasyonların doğruluğunu artırır. Simülasyonlar daha gerçekçi bilgileri almak için, gerçek sistemlerle etkileşmelidir. Gerçek dünya varlıklarının simülasyona katılması değişik senaryolarla yapılabilir. İnsan ve gerçek dünya sistemlerinin *simülasyona katılım derecesine göre* aşağıdaki simülasyon senaryoları vardır [156] :

- Live simulation: İnsan operatörler, gerçek dünya sistemlerini işletmek için simülasyona dahil edilirler. Zaman süreklidir. Örnek: gerçek bir araç aküsünü elektrik test aracı ile test etmek.
- Virtual simulation: İnsanlar tarafından işletilen simüle edilmiş sistemler içerir. Zaman ayrıktır (discrete). İnsan merkez roldedir. Örnek: uçuş simülatörü.
- Constructive simulation: Klasik bilgisayarlı simülasyondur. Simüle edilmiş sistemleri kullanan simüle edilmiş insanlar içerir. İnsan operatörler sadece sisteme bilgi girişi sağlarlar ve simülasyon sonucuna etkileri yoktur.

Bu üç kategori arasında hatlar keskin çizgilerle tanımlı değildir [157] ve simüle edilmiş varlıkların gerçek dünya cihazları ile etkileşimini içeren bir kategori eksiktir.

Zaten mevcut simülasyon araçlarında da bu destek eksiktir. Hali hazırdaki araçların hiçbirisi bu tür bir karışımı (gerçek zaman ve simülasyon zaman varlıkları bir arada kullanma) tam anlamıyla desteklememektedir. Literatürdeki çalışmalardan ancak bir kaç tanesi, gerçek ve simülasyon zaman varlıklarını birleştirmeye odaklanır. SimJava [158], Java RMI (Remote Method Invocation) arayüzleri üzerinden birbirleriyle haberleşen bileşenlere sahip dağıtık simülasyonlara olanak sağlar. Simülasyon bileşenleri, RMI arayüzü sağlayan gerçek dünya yazılımları ile de haberleşebilirler. Ancak dış yazılımlar bir simülasyon varlığı gibi davranamaz.

Diğer bir çalışma [159] ise olay-tabanlı DTS yaklaşımına göre çalışan dağıtık bir simülasyon ve bilgi sağlama mimarisi önerir. Gerçek dünya uygulamalarının paralel olarak simülasyon oturumu ile etkileşimlerine olanak sağlar. Simülasyon varlıkları, uzaktaki istemci uygulamalara, simüle edilen sistemin görünüşünü sağlarlar. Bu model, dış kullanıcılara, ya pasif (simülasyonu izler ve sonucuna etki etmez) ya da aktif (simülasyon ile etkileşip çıktısına etki eder) olarak simülasyonla etkileşme olanağı sağlar ama simülasyonun var olan gerçek dünya uygulamalarına ulaşması eksiktir. Kodları değiştirilmiş dış istemciler simülasyonun çıktılarını canlı izleyebilir ve GUI'ler üzerinden simülasyon ile etkileşebilir. Simülasyon servisi, dış sistemlere, sadece simülasyona ilişkin veri almak üzere erişir. Bu modelin de uzak servisler ile zaman senkronizasyonu problemi vardır.

Bu tez çalışmasında, bir *hibrit simülasyon modeli* geliştirilmiştir. HeteroSim [9-11] adını verdiğimiz java tabanlı ve ayrık olay (discrete event) yaklaşımıyla çalışan bu model, hem gerçek dünya yazılımlarını hem de simülasyon kodlarını bir arada kullanmaya olanak sağlar. Bu heterojen simülasyon oturumlarında, hem gerçek dünya varlıklarından ve insanlardan simülasyon varlıklarına doğru hem de tersi yönde bir etkileşim başlatmak mümkün olur. Dahası modelimizde, simülasyonda kullanılacak gerçek dünya yazılımının kodunda herhangi bir değişiklik gerekmez.

Bu modelin kavram ispatı için geliştirilen prototip, gerçek dünya varlıkları ile JMS [97] ve JMX [160] iletişim arayüzü üzerinden haberleşir. Tezin özgün katkısı olan bu model, Bölüm 5.3'de detaylı tanıtılmıştır.

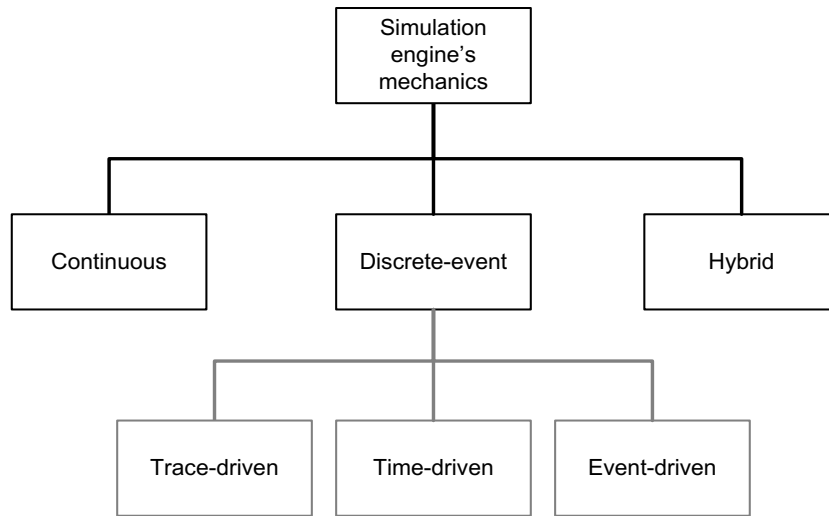
Bu yetenek sayesinde mesela, hazır yazılımları doğrudan bir simülasyona dahil edip hızlıca heterojen simülasyonlar hazırlanabilir. Örneğin, yazılım testi amacıyla HeteroSim kullanılmak istenirse, test edilecek olan yazılım parçasını gerçek dünya

varlığı olarak, test senaryolarını ise simülasyon varlığı olarak gerçeklemek mümkündür. Böylece, simülasyon aracı içinde test senaryolarını gerçeklemek mümkün olur. Kendisine, senaryo tabanlı yazılım testi ve simülatör geliştirme gibi pek çok uygulama alanı bulabilecek olan HeteroSim, Bölüm 4.3’de tanımlanan evrensel YBH senaryolarını gerçeklemek için kullanılmıştır. Bu kullanımda, gerçekleşmiş olan Police PBM yazılımı gerçek dünya yazılımlarını, simüle edilen YBH bileşenleri ise simülasyon varlıklarını temsil eder. Böylece bu YBH bileşenleri simülasyonunda, politikalar kullanıp etkilerini inceleyebilme olanağı elde edilmiş olur.

Bu bölüm, önce bir temel simülasyon bilgisi daha sonra da sırasıyla HeteroSim modeli ve Police ile kullanımı konularını içerir.

5.2 Simülasyon Araçları

Simülasyon kavramı gerçek dünya sistemlerini modelleme ve değerlendirmek suretiyle davranışlarını anlamak için kullanılmaktadır. Simülasyon teknolojileri genellikle ayırık olay tabanlı (*discrete event-driven*) programlar olarak geliştirilir. Olaylar (event), zaman etiketleri taşıyan mesajlar olarak ele alınır ve simülasyon saati ilerledikçe zamanda sıralı olarak işlenirler [161]. Simülasyon modellerini gerçekleştirme konusundaki kavramlar Buyya tarafından [162] makalesinde açıklanmış ve simülasyon araçlarının detaylı bir sınıflandırması verilmiştir (Şekil 5.1). Bu bölümde, bu sınıflandırmadan, tezimizi ilgilendiren kriterlere göre olanları vurgulanmıştır.



Şekil 5.1 : Simülasyon motoru mekaniğine göre sınıflandırma [162].

Her simülatör, simülasyon modelini yürüten bir simülasyon motoruna sahiptir. Zamana göre olan davranışlarına göre üç çeşit motor vardır [162]:

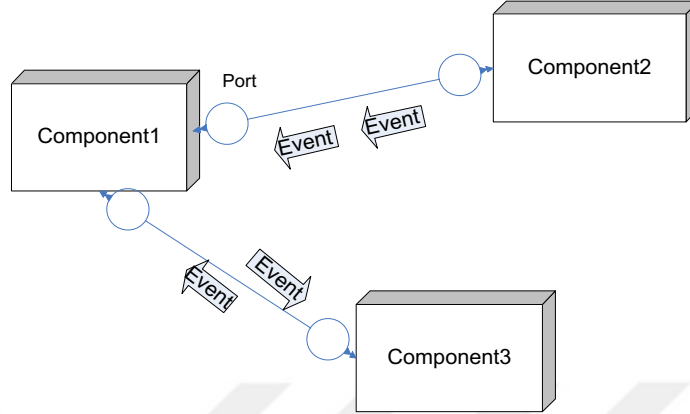
- Sürekli zaman (continuous time) simülasyon: Durum değişimi sürekli zamandadır. Bu tür sistemler genelde diferansiyel denklemlerle tanımlanır ve çözülür.
- Ayrık olay (discrete event/time simülasyon -DES/DTS-) simülasyon: Durum değişimleri sadece belirli zaman anlarında (*observation points*) oluşur. Sistem durumu sadece bu zaman noktalarında değerlendirilir. Bu noktaların arası kıaldıkça sürekli zaman simülasyona yaklaşılr. DTS, *trace-driven*, *time-driven* ve *event-driven* olmak üzere üç alt sınıfa bölünür. Trace-driven DTS, daha önce başka bir ortamdan toplanmış olay kayıtlarının yürütür. Time-driven DTS, sabit zaman dilimlerinde ilerler. Event-driven DTS ise düzensiz zaman aralıklarında ilerler ve olayların rasgele olduğu sistemleri modellemede işe yarar.
- Hibrit (Continuous time&discrete event) simülasyon: Zaman soyut olarak sürekli ve gözlem periyodu gerçek zaman aralıklarıdır (genelde basitlik için saat sıfırdan başlar). Operasyon akışı, olay zamanları dizisi ile belirlenir (eşit veya değişken aralıklar) ve sistem durumundaki ayrık değişiklikler, bu zaman anlarında gerçekleşir (olayların etkileşimi). Peşpeşe oluşan olaylar arasında sistem durumu sürekli zamanda değişir.

5.2.1 Literatürde simülasyon araçları

Geliştirilmiş çok sayıda simülasyon dili, simülasyon ortamı (Parsec [163]), simülasyon kütüphanesi (SimJava [164]) ve uygulamaya özel simülatör (OMNET++ network simulator [165]) vardır. Daha sonra, OMNET++[165], J-Sim [166], JavaSim[167] gibi temel simülasyon araçları kullanılarak bir üst seviye simülasyon araçları geliştirilmiştir. Bu alana özgü araç örnekleri, GridSim[168], SimGrid [15, 169, 170] ve MicroGrid [171]'dir. Java dili, çok-lifli (multi-threaded) programlama ve iş sıralama konusunda doğasından gelen bir destek içerdiği için pek çok simülasyon aracı Java dili ile geliştirilmiştir [172].

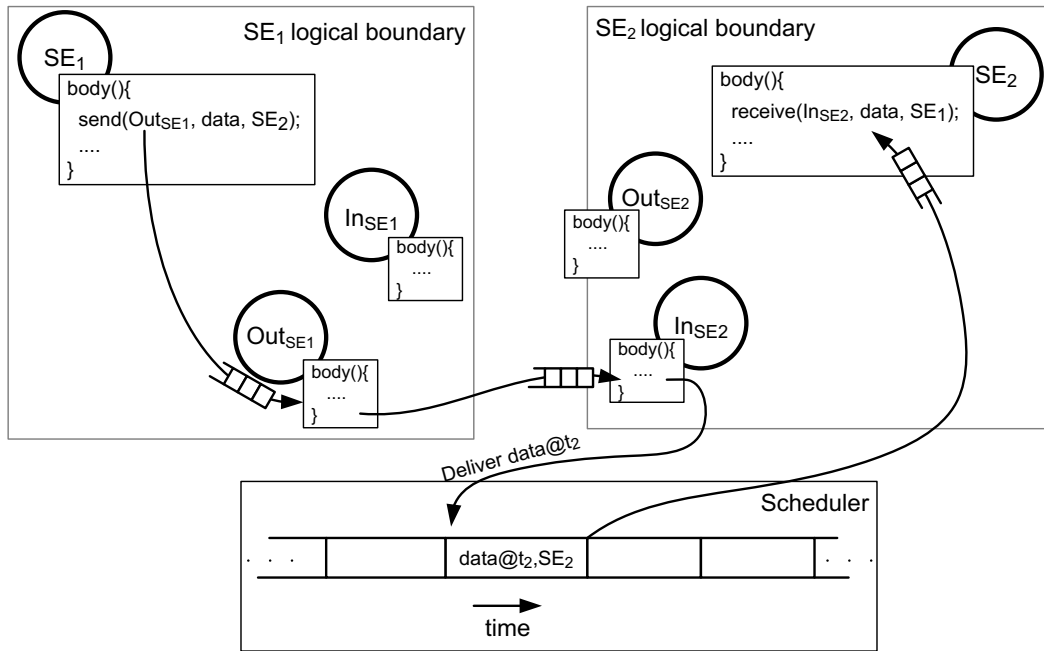
JSIM [173], Web-tabanlı simülasyonu da destekleyen bir diğer Java tabanlı simülasyon ve animasyon ortamıdır. Simülasyonlar olay veya süreç-tabanlı yaklaşımla oluşturulabilir. J-Sim (JavaSim), açık kaynak kodlu, süreç-tabanlı DTS aracıdır [166]. Olay-tabanlı ayrık simülasyonları destekler. Şekil 5.2'de gösterildiği üzere, J-Sim

komponentleri gevşek bağlıdır (portlarının birbirine bağlanması ile etkileşirler). Portlar komponentlerin dış dünya ile tek etkileşme noktalarıdır. Komponentler asenkron olarak aynı anda farklı veriler üzerinde çalışabilir. J-Sim hem DTS modunda hem de gerçek-zamanlı süreç-tabanlı simülasyon modunda çalışabilir.



Şekil 5.2 : Simülasyon elemanları.

SimJava [164, 174] ve Silk [175], süreç-tabanlı DTS için geliştirilmiş ilk Java kütüphanelerindendir. SimJava, aslında, HASE++ (Hierarchical computer Architecture design and Simülasyon Environment) adlı C++ kütüphanesinin Java versiyonudur. SimJava simülasyonları, kendi Thread'i içinde paralel olarak çalışan bir dizi simülasyon varlıklarından oluşur. J-Sim'deki gibi SE'ler portlarından birbirlerine bağlıdır ve bu portlar üzerinden olay alışverişi ile etkileşirler. Olay kuyruğunu yöneten bir Zamanlayıcı, simülasyon saatini ilerletir (Şekil 5.3).



Şekil 5.3 : SimJava varlıkları arasında zamana bağlı olay dağıtımı [168].

SimJava'nın ardışıl DTS algoritması şöyle çalışır: Zamanlayıcı (Scheduler) gelecekteki olayların zaman sıralı listesini tutar (FEL: Future Event List). SE'ler başlatılır ve RUN durumuna sokulur. Bir SE, herhangi bir simülasyon fonksiyonunu çağırdığında, Zamanlayıcı, bu SE'ye ait Thread'i durdurur ve çağrılan fonksiyonunun işlendiğini göstermek üzere FEL'e bir olay kaydı ekler. Tüm SE'ler durduğunda, Zamanlayıcı, FEL'den sıradaki olayı çeker, simülasyon saatini bu olayın saatine iletir ve uygun SE'yi tekrar başlatır. Bu işlem FEL'de olay kalmayınca kadar tekrarlanır. SimJava kullanılarak geliştirilen GridSim [120], bazı Grid varlıklarını modelleyebilir. SE'ler arası bilgi akışı SE'lere ait Input/Output Varlıkları üzerinden de gerçekleşebilir. Olaylar (Internal/External), hemen veya belirtilen süre sonrasında alıcı SE'lere dağıtılabilir. Olaylar senkron veya asenkron işlenebilir. EXTERNAL olaylar senkron veya asenkron olabilirken INTERNAL olanlar, ölümcül kilitlenmeyi engellemek için sadece asenkron olabilir.

5.2.2 YBH amaçlı simülasyon araçları literatür incelemesi

Bu bölümde, YBH sistemlerinin simülasyonu için kullanılan araçlar incelenmiştir. Mikroişlemci ve ağ protokolü tasarımı alanlarında uzun geçmişi olan simülasyonlar, YBH alanında nispeten yeni sayılabilir. Kullanılan mevcut simülatörler ise daha ziyade alt tiplere (Grid, HPC, MPP, cluster vb.) özgü geliştirilmiştir. YBH simülasyon araçları, [15] makalesinde aşağıdaki kriterlere göre incelenmiştir:

- (i) *Simülasyon modeli*: Kaynakları ve bunları kullanan uygulama aktivitelerinin modellenmesi yöntemidir. Literatürde *İşlemci*, *Şebeke*, *Veri depolama* tiplerinde kaynaklar modellenir.
 - *CPU modellemesi*: İşlemcinin saat-bazında modellenmesidir ve nadiren kullanılır [176]. Çalışırken çok işlemci gücü tüketir ve simülasyon modelinin tanımlanması zordur. Simülatörlerin çoğu, hesaplama süresi için analitik model kullanır ($iş_i\_yürütme\_süresi = işin\_boyu / işlem\_hızı$).
 - *Disk modellemesi*: Çoğu simülatör, sabit/rassal arama süresi ve sabit veri transfer hızı parametrelerine dayalı, basit analitik model kullanır. Doğruluğu yüksek disk modellemesi zor olduğundan bu tip simülatör sayısı azdır. Örnek: DiskSim [177], OMNet++ tabanlı SIMCAN simülatörü.

- *Ağ modellemesi:* Ağ iletişiminin, paket seviyesinde detaylı modellenmesi gerekir. Bunun yerine çoğu simülatör, sadece ağ gecikmesini içeren basit analitik model kullanır (*gecikme + veri_boyu/bandgenişliği_payı*). GridSim [168], CloudSim [28] ve bunlardan türemiş diğer simülatörler [178-180] basit bir ağ modellemesi kullanır. Bu gibi simülatörlerin ağ modellemelerinin doğruluğu [181] çalışmasında incelenmiştir.
- (ii) Platform tanımlama: Simüle edilecek YBH platformunun topolojisini tanımlama yöntemidir. CPU tipli kaynaklar genellikle hız parametresi (FLOPS veya MIPS) ile tanımlanır Disk tipli kaynaklar ise iki parametre ile (seek time ve I/O bandwidth) tanımlanır. Ağ kaynakları da iki parametre ile (link latency ve link bandwidth) tanımlanır.
- (iii) *Uygulamayı tanımlama:* Simüle edilecek aktiviteleri tanımlama yöntemidir. Temel olarak üç yaklaşım vardır:
- Off-line simülasyon, uygulamanın gerçek platformda çalışmasından elde edilen iz kayıtlarının simülasyonda tekrar oynatılmasıdır. Özellikle MPI simülasyonu için kullanılır.
 - Formal tanımlama, uygulamanın gerçekleştirilmesi yerine, uygulamanın işleyişinin formal tanımı ile yapılır (PeerSim ve SimBA). Daha küçük boyutlu ve ölçeklenebilir olsa da karmaşık uygulamaları formal olarak tanımlamak zor olabilir.
 - Programlamayla tanımlama, simüle edilecek uygulamanın kullanıcı tarafından CSP (Concurrent Sequential Processes) olarak kodlanmasına dayanır. Örnek: HeteroSim, GridSim ve SimGrid.

YBH simülasyonu için çeşitli simülatörler geliştirilmiştir. Şekil 5.4’de bilinen önemli simülatörlerin özellikleri karşılaştırmalı olarak özetlenmiştir ([15] makalesinde bulunan tabloya HeteroSim için bir satır eklenerek tablo yeniden hazırlanmıştır).

Simulator	Community of Origin					Application			Network						CPU		Disk		
	high performance computing	grid computing	cloud computing	volunteer computing	peer-to-peer computing	execution trace	abstract specification	programmatic specification	latency	bandwidth	store-and-forward	ad hoc flow	TCP flow-level model	packet-level	scaled measured delay	scaled user-provided delay	capacity	seek + transfer	block-level
PSINS	✓					✓			✓	✓		✓			✓	✓			
LogGOPSim	✓					✓			✓	✓					✓	✓			
BigSim	✓								✓	✓				✓	✓	✓			
MPI-SIM	✓							✓	✓	✓					✓	✓			
OptorSim		✓						✓	✓	✓		✓				✓	✓	✓	
GridSim		✓						✓	✓	✓	✓	✓				✓	✓	✓	
SimGrid		✓				✓	✓	✓	✓	✓			✓	✓	✓	✓	✓	✓	
GroudSim		✓	✓					✓	✓	✓		✓				✓			
CloudSim			✓					✓	✓	✓	✓					✓	✓	✓	
iCanCloud			✓					✓	✓	✓				✓		✓	✓	✓	✓
SimBA				✓			✓	✓	✓	✓						✓			
EmBOINC				✓			✓	✓	✓	✓						✓			
SimBOINC				✓			✓	✓	✓	✓			✓			✓			
PeerSim					✓		✓		✓	✓									
OverSim					✓			✓	✓	✓				✓					
HeteroSim	✓	✓			✓	✓		✓	✓	✓	✓					✓			

Şekil 5.4 : HeteroSim’in simülatörler ile karşılaştırılması¹.

HeteroSim’in tablodaki kriterlere göre özellikleri şunlardır:

- YBH alanlarını hedefler. *Peer-to-peer* hesaplama için kaynak modellemesi (her düğüm kendi scheduler’ını içerecek şekilde) ve iş yükü yönetimi yapılabilir. Bulut hesaplama için uygun değildir. Sanal kaynak modellemesi yapılamaz. *Volunteer* hesaplama için de uygun değildir. İş yüklerini kaynakların kendilerinin merkezi kuyruktan çekmesi, HeteroSim’in mevcut hali ile modellenemez. Ancak destekler hale getirmek zor değildir.
- Yürütme izleri tutulmaktadır.
- Detaylı (paket seviyesi) şebeke modellemesi yapılmaz. GridSim’den miras alınan, simülasyon varlıkları arasındaki kanallara gecikme konulabilir.
- İşlemci modelleme: kullanıcı tarafından sağlanan kaynakların işlemci hızları nedeniyle “*scaled user provided delay*” ile işlemci modellenir.
- Disk modelleme desteği yoktur. Sadece hesaplama kaynağı modelleyebilir.

Bu simülatörler için, [15] makalesinde verilen tanımlarından ve literatür araştırmasından elde ettiğimiz bilgilerin özeti aşağıdaki maddelerde verilmiştir:

¹ [15] makalesinden alınarak ek satır ile genişletilmiştir.

A. *HPC* için geliştirilen simülatörler,

MPI (Message Passing Interface) uygulamaları için iki yaklaşım kullanılır:

- *off-line simülasyon*: Uygulamanın gerçek sistem üzerindeki çalışma izlerinin simülatörde yeniden oynatılmasıdır (Örnekler [182, 183] makalelerinde tanıtılmıştır).
- *on-line simülasyon*: Gerçek uygulama kodu, hedef platformu taklit eden bir platform üzerinde çalıştırılır (Örnek [176] makalesi).

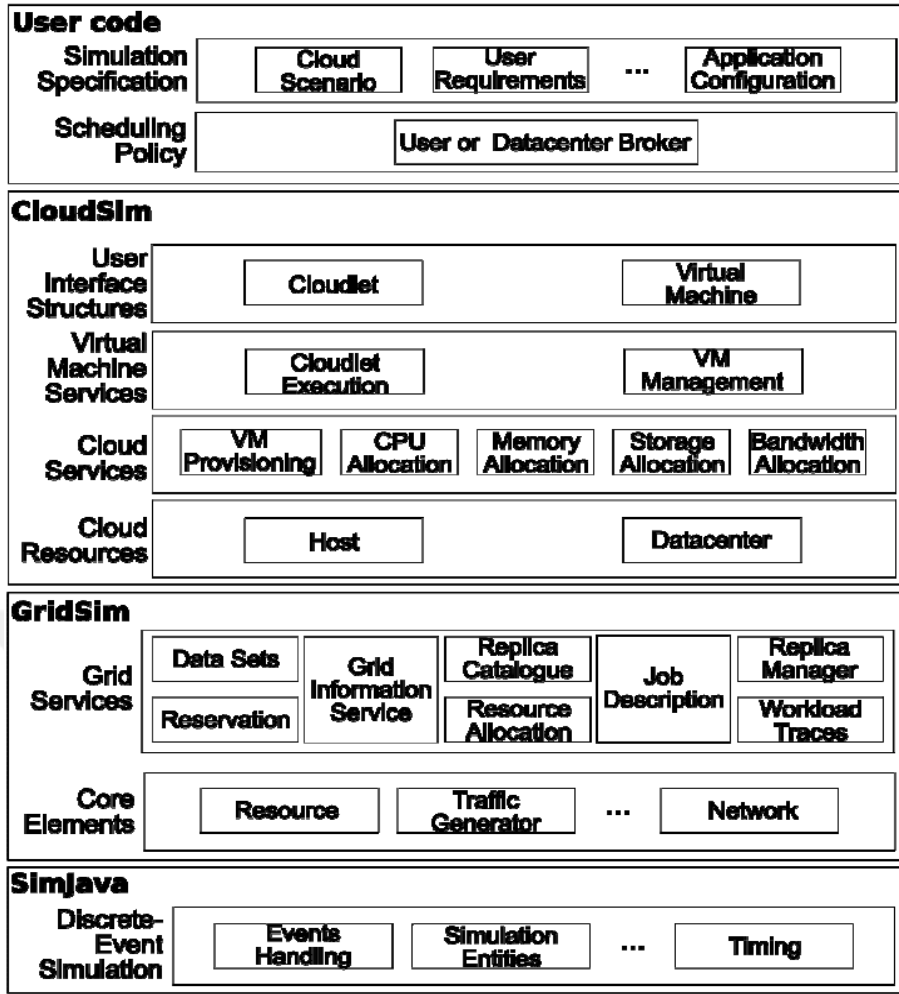
B. *Bulut hesaplama* için geliştirilen simülatörler,

Mevcut Grid/cluster simülatörler, sanal kaynakları modelleme desteği olmadığı için bulut uzayında doğrudan kullanılamazlar. Bulut simülatörlerinin (CloudSim, SPECI, CDOSIM ve DCSim) karşılaştırması [184] makalesinde verilmiştir. iCanCloud [185] sadece Bulut sistemleri için geliştirilmiştir. CloudSim [28], GridSim iskeleti üzerine kurulmuş olan en gelişmiş bulut simülatörüdür (Şekil 5.5). Sanal makina, federe bulutlar, veri merkezi modellenebilir. TeachCloud ve DynamicCloudSim [186] gibi pek çok türevi vardır. CloudSim, SimJava DES motoru üstünde GridSim motorunu kullanır. CloudSim, bulutların sanallaştırma özelliğinden dolayı istemci uygulamalarına kaynak atamasını, Grid simülatörlerinden farklı yapar. Uygulamalar VM (Virtual Machine)'lere, VM'ler ise hesaplama düğümlerine atanır. Kaynak modellemesi, DataCenters\Hosts\VMs\Uygulamalar hiyerarşisi ile yapılabilir. Bir makina birden fazla VM tarafından, bir VM ise birden fazla servis uygulaması tarafından aynı anda paylaşılabilir.

GroudSim [184] Grid ve Cloud simülasyonu için kullanılabilen, Java dilinde geliştirilmiş, DES tabanlı simülatördür (IaaS desteği sağlar). Olay tabanlı yürütme modeli ile ölçeklenebilirlik sağlar.

C. *Volunteer computing* için geliştirilen simülatörler,

Bu alandaki simülatörler, BOINC adlı *volunteer computing* platformunun fonksiyonelliklerini simüle etmeye çalışır. Örneğin SimBA [187], BOINC istemcilerini modeller. EmBOINC [188] gerçek BOINC kodunu kullanarak BOINC sunucusunu modeller. SimBOINC [189] ise bütün BOINC sistemini simüle etmek için BOINC kodu ile SimGrid kodunu birleştirir.



Şekil 5.5 : CloudSim mimarisi [28].

D. *Peer-to-Peer computing* için geliştirilen simülatörler,

Bu simülatörlerinin ([190]'de özetlenmiştir) çoğunda, milyonlarca bilgisayar içeren simülasyonlar çalıştırmak için doğruluktan ödün verilir. Örneğin, karmaşık ağ modellemesi kullanılmaz. PeerSim [191] en çok kullanılan, basit ama ölçeklenebilir simülatördür. OverSim [192] ise OMNeT++[165] üzerine inşaa edildiği için daha gerçekçi ağ simülasyonları için kullanılabilir. Diğer örnekler, P2PSim [193] ve PlanetSim [194]'dir. P2PSim de bir DES simülatördür ve çok sayıda peer-to-peer protokol gerçeklemesi içerir (*Koorde*, *Kademlia* vb.).

E. *Küme ve Grid computing* için geliştirilen simülatörler,

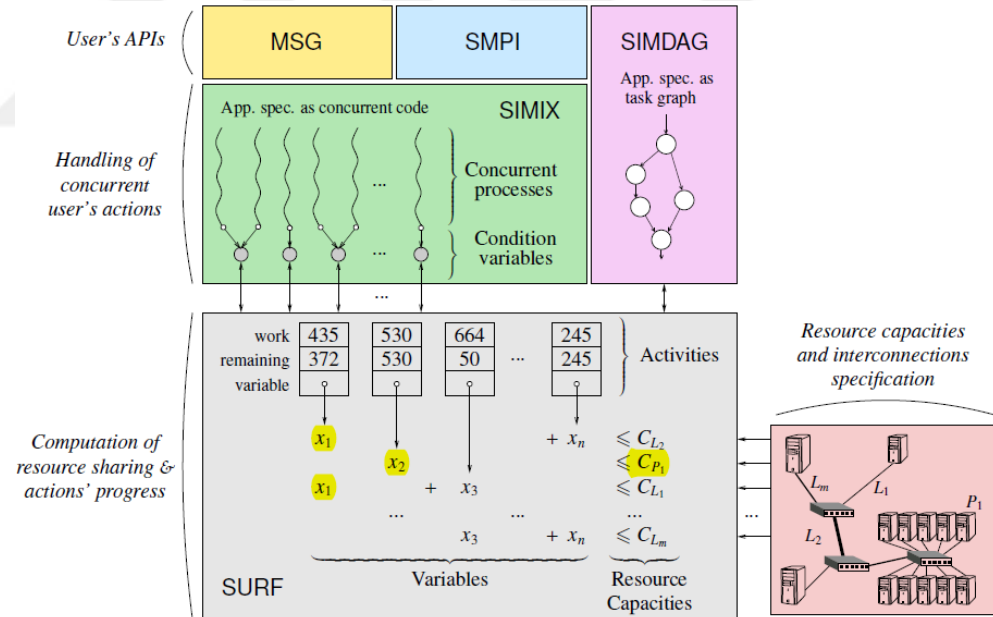
ChicSim (Chicago Simulator) [195], modüler ve genişleyebilir DES tabanlı, Data Grid simülasyon sistemi olmakla birlikte geliştirmesi durmuştur.

GangSim, [120] VO-Centric Garglia (DES tabanlı simulator) [119], VO ve kaynak modellemesine olanak veren, diğer bir Grid simülatörüdür. Garglia izleme sisteminden türetilmiştir.

[106]'da tanıtılan Küme simülatörü, Maui iş sıralayıcı içerir. Maui'nin simulasyon modu vardır. Bir kaynak dosyası ve bir iş yükü izi dosyasını girdi olarak alır ve iş sıralayıcının, işleri planlamasını simüle eder.

GridSim [168] ve SimGrid [15] simülatörleri geniş bir kullanıma sahiptir. GridSim, SimJava üzerine inşaa edilmiş olan DES tabanlı bir grid simulasyon aracıdır. Her bir *cluster*, heterojen bir simülasyon varlığı olarak modellenir. Diğer varlıklar, kullanıcılar, makinalar ve ağ iletişimidir. GridSim, ekonomi tabanlı detaylı kaynak tahsis modülleri ile öne çıkar. Bu araç ile VO tabanlı sınırlı bir grid modellemesi de yapılabilir.

SimGrid [15], C dilinde geliştirilmiş DES tabanlı bir simülasyon altyapısıdır. (Şekil 5.6). SimGrid, dağıtık uygulamaları "Grid platformlarında çalıştırma simülasyonu için geliştirilen bir platformdur.



Şekil 5.6 : SimGrid mimarisi [15].

SimGrid, simülasyon geliştiricileri için üç tip API sağlar:

- MSG, simüle edilecek uygulamayı bir dizi eş zamanlı proses olarak tanımlamaya izin verir. Bu Prosesler, (C, C++, Java, Lua, Ruby) MSG çağrıları yaparak aktiviteleri (hesaplama/haberleşme) simüle ederler.

- SMPI, simüle edilecek uygulama, MPI tabanlı mevcut bir uygulamadan (kodunu değiştirmeden) otomatik olarak üretilen bir dizi proses olarak tanımlanır. MSG ve SMPI üzerinden çalışan proseslerin simülasyonu, SIMIX modülü içinde çalışır. SIMIX içinde çalışan prosesler, *Koşul Değişkenleri* üzerinden senkronize olurlar. Her koşul değişkeni, simüle edilen bir aktiviteye (hesaplama/transfer) karşı düşer ve prosesin, aktivite tamamlanıncaya kadar bekletilmesini sağlar (Şekil 5.6).
 - SIMDAG, kullanıcının uygulamasının, eş zamanlı proses yerine haberleşen görevler (task) grafi olarak tanımlamasını sağlar (non-cyclic). SURF (Simülasyon Core) aktivitelerin kaynaklar üzerinde çalışmasını simüle eden modüldür. Her aktivite, toplam ve kalan iş miktarı ile modellenir (İşlemci çevrim sayısı, transfer edilecek byte sayısı vb.) Aktivite tamamlanınca ilgili SIMIX koşul değişkeni veya SimDIAG'da ilgili task'ın blokesi kaldırılır. İşlemci ve depolama kaynaklarını birbirine bağlayan ağın topolojisi de modellenebilir. Simülasyonun hesaplama yoğun kısmı, simülasyon modelinin değerlendirilmesi değil, simüle edilen proseslerin yürütülmesidir. SimGrid, simülasyon modelinin hesaplamasını, tek bir thread ile yürütür (SURF içinde). Böylece bağlam anahtarlama sayısı azalır.
- GES (Grid Economics Simulator) [196], tek lifli (thread) bir simülatör örneğidir. Birden fazla kaynak tahsis algoritmasını destekler. Yeni iş sıralama algoritmaları ile genişletilebilir. *Distribution Layer* katmanı sayesinde, simülasyonlar birden fazla bilgisayar üzerinde çalıştırılabilir.

5.3 HeteroSim

Tez kapsamında, Bölüm 4.3'deki evrensel YBH modeline göre simülasyonlar gerçekleyebilen ve Bölüm 5.1'de açıklanmış olan ihtiyaçlara cevap veren, simülasyon yazılımlarını gerçek yazılımlarla birlikte çalıştırabilen, HeteroSim [10, 11] adını verdiğimiz bir hibrit simülasyon ortamı geliştirilmiştir. Bölüm 5.2'deki literatür araştırmasında incelediğimiz ve genişletmeye müsait yapıya sahip olan GridSim ve JavaSim, HeteroSim'in yapı taşı olarak seçilmiştir. Süreç-tabanlı bir DTS olan HeteroSim ile elde edilen yenilikler aşağıda listelenmiştir:

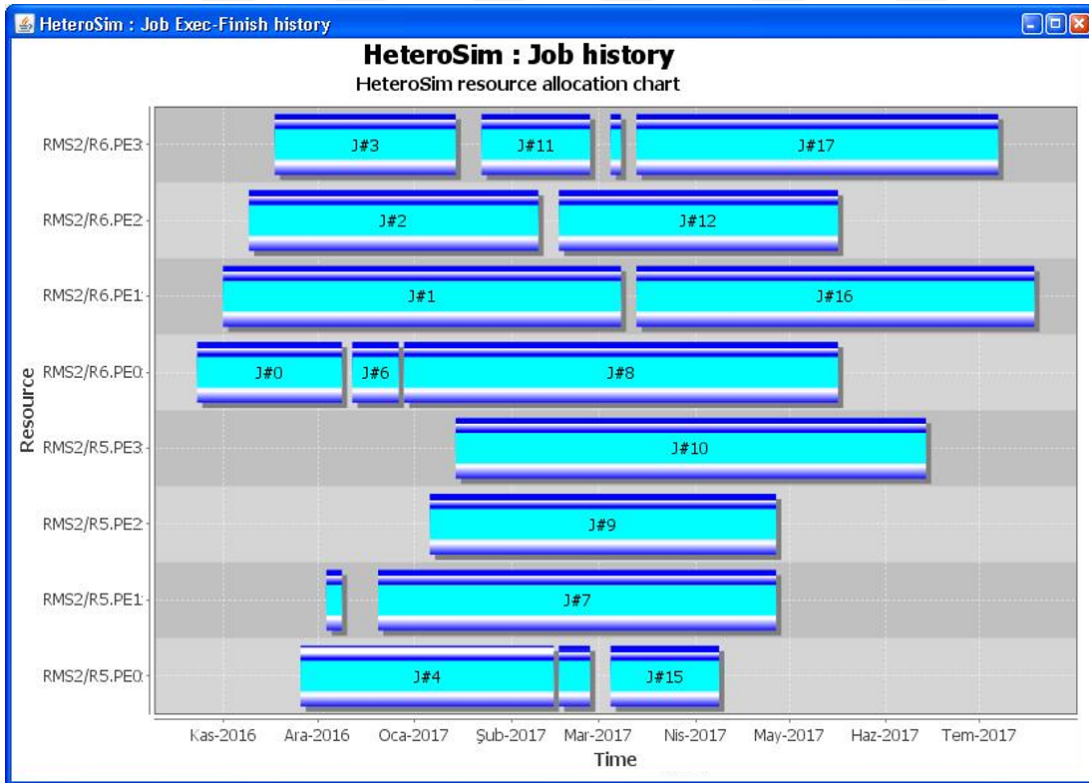
- GridSim ve JavaSim’de olmayan, simülasyon varlıklarının kendi arasında ve simülasyon dışı varlıklarla etkileşimleri için arayüzler sağlanmıştır. Simülasyon ve gerçek yazılımları içeren heterojen simülasyonlar yapılabilir.
- Daha yüksek çözünürlükte kaynak modellemesi mümkündür. Zira GridSim’de yapı taşı olarak bir iş sıralayıcı (scheduler) tarafından yönetilen bir grup bilgisayar kullanılır. HeteroSim’de ise her bir makine ve üzerindeki işlemciler (Processing Element) en küçük yapı taşıdır.
- Simülasyonlarda PEP ile etkileşim arayüzü üzerinden politika uygulama eylemleri yapılabilir. Simülasyondaki RMS’in bütün bileşenleri yönetilebilir.
- Simülasyonda, yerel ve uzaktan gelen işler arasında ayırım yapılabilir.
- Simülasyona Trader, RMS, Dispatcher ve LIS (Local Information Service) isimli yeni varlık tipleri eklenmiştir. Bu sayede, GridSim’de Scheduler’a toplanmış fonksiyonellikler, olması gereken bileşenlere dağıtılabilmektedir.
- Simülasyon başlamadan önce RMS topolojisinin otomatik olarak oluşturulması sağlanmıştır.
- İki den fazla derinlikte RMS hiyerarşisi inşaa etmek mümkün olmuştur.
- Simülasyonların hızını ayarlama yeteneği eklenmiştir.
- Simülasyon senaryosunun tanımlanacak noktalarında, simülasyondaki varlıklar arasında senkronizasyon/randev-vous gerçekleştirme desteği eklenmiştir.
- İşin, farklı kaynaklara ait hesaplama kaynaklarına (Processing Element) atanabilmesine destek eklenmiştir. Dahası, bir iş farklı RMS’lerin kontrolündeki birden çok kaynağa atanabilmektedir.
- Kaynak tahsisi için, politikaların içinde tetiklenebilen modlar eklenmiştir (*LONGEST_FIRST*, *LOCAL_FIRST*, *EQUAL*, *REMOTE_FIRST* ve *LEAST_REMAINING_FIRST*).
- Kaynakları isimlendirmek ve adreslemek için bir şema eklenmiştir (Örnek: YBH1\RMS2\Resource6).
- Kaynak ve Kullanıcıların VO (Virtual Organization) üyesi olması mümkündür.
- *ResourceLoader* isimli bileşen sayesinde YBH merkezlerine ait iş yükü iz dosyalarının (workload traces) kullanılabilmesi sağlanmıştır.
- YBH, RMS ve Kullanıcılar bazında çeşitli metrikleri yürütme anında canlı hesaplayan mekanizmalar gerçekleştirilmiştir. Bu metrikler Çizelge 5.1’de

verilmiştir. Modüler mimari sayesinde, yeni metrikler kolayca eklenebilmektedir.

Çizelge 5.1 : HeteroSim metrik listesi.

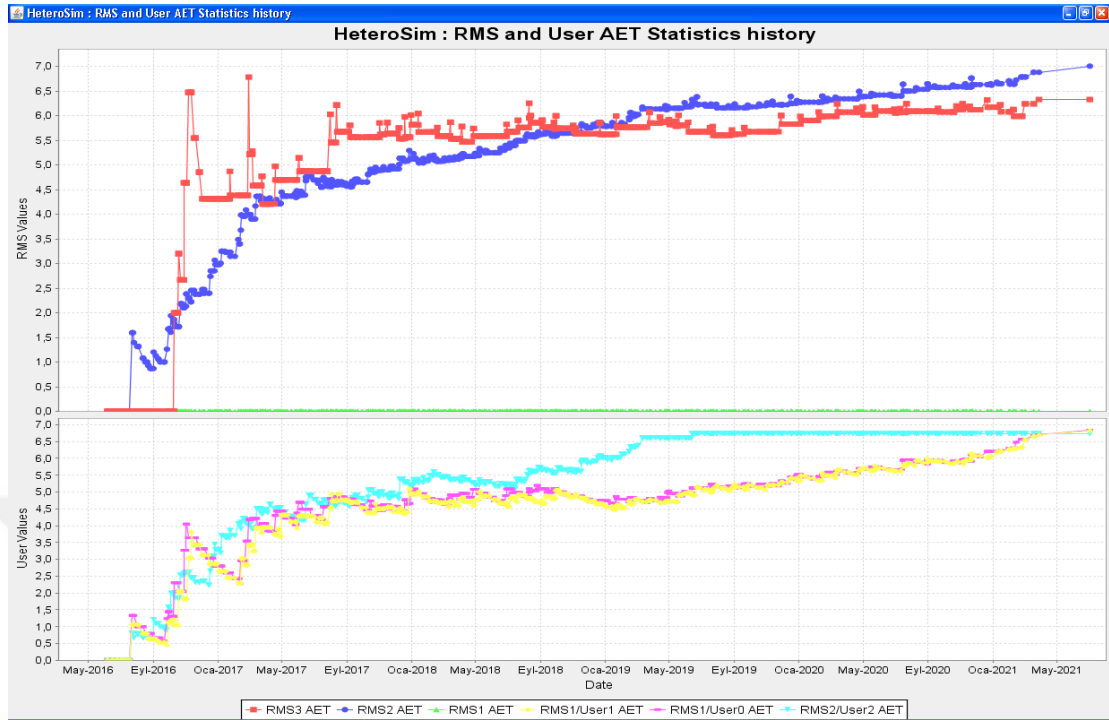
Metrik	Çözünürlük	RMS	Kullanıcı	YBH
AET (Average Execution Time)		√	√	√
AWT (Average Waiting Time)		√	√	√
AJL (Average Job Length)		√	√	√
TJL (Cumulative Total Job Length)			√	
MakeSpan				√
Max, Min, Average Job length				√
System workload Ratio				√

- Eklenen bu metriklerin, politikaların kısıt bölümlerinde kullanılabilmesi de sağlanmıştır.
- Her bir metrik için simülasyon sonucunda yürütme sonuçlarının, JFreeChart [197] ile geliştirdiğimiz grafiksel arayüzlerde gösterilebilmesi sağlanmıştır.
- *ResourceAllocation* grafiği ile işlerin kaynaklar üzerinde çalışma geçmişleri kronolojik olarak gösterilebilir. Örnek arayüzler Şekil 5.7’de gösterilmiştir.



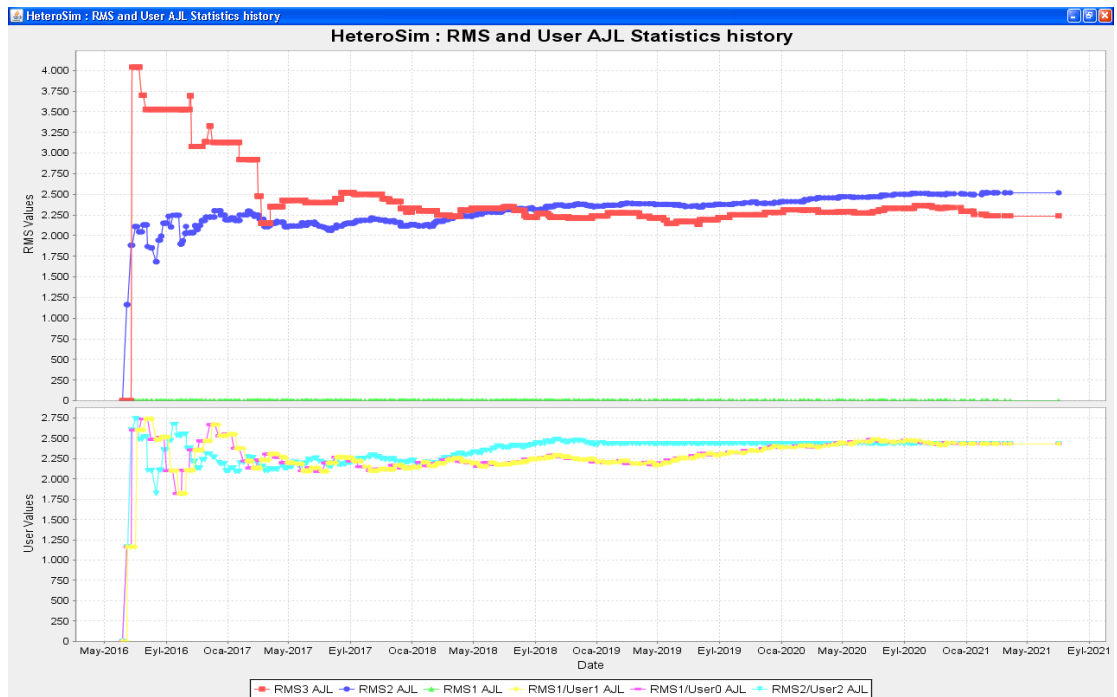
Şekil 5.7 : HeteroSim kaynaklara atama grafiği.

- Şekil 5.8’de gösterilen AET (Ortalama Yürütme Süresi) metriği grafiği ise hem sistem hem de kullanıcı bazında görselleştirme yapabilmektedir.

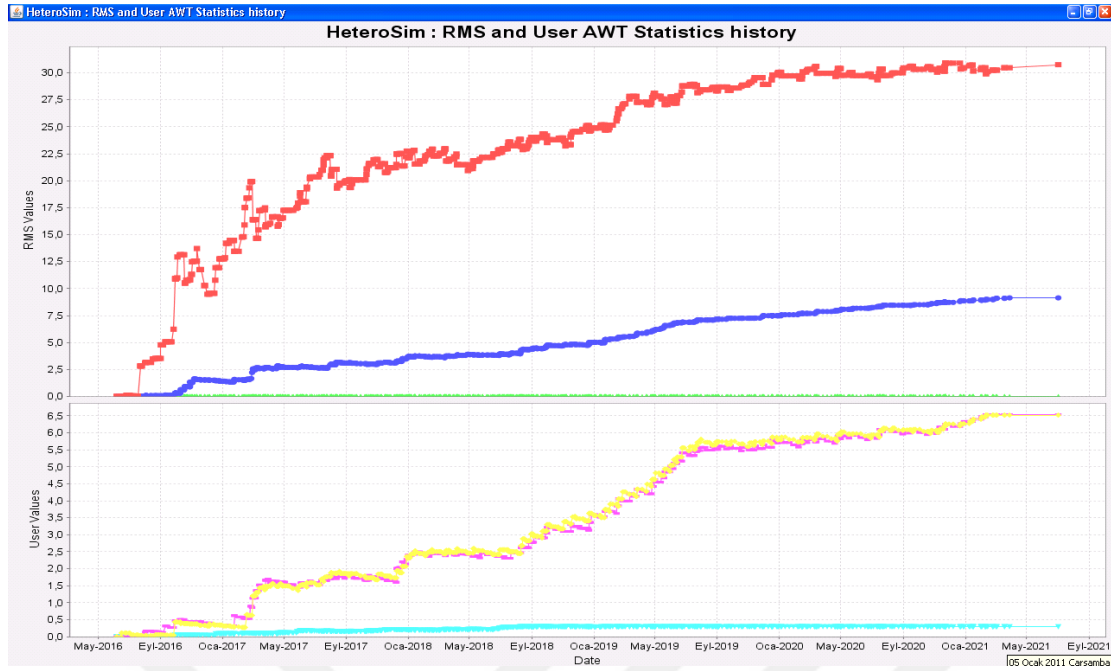


Şekil 5.8 : HeteroSim AET metrik grafiği.

- AJT (Ortalama İş Yüğü Uzunluğu) metriği grafiği ise Şekil 5.9’da gösterilmiştir. Benzer olarak hem sistem hem de kullanıcı bazında görselleştirme yapabilmektedir. Şekil 5.10 ise AWT metriği enkranını gösterir.



Şekil 5.9 : HeteroSim AJL metrik grafiği.



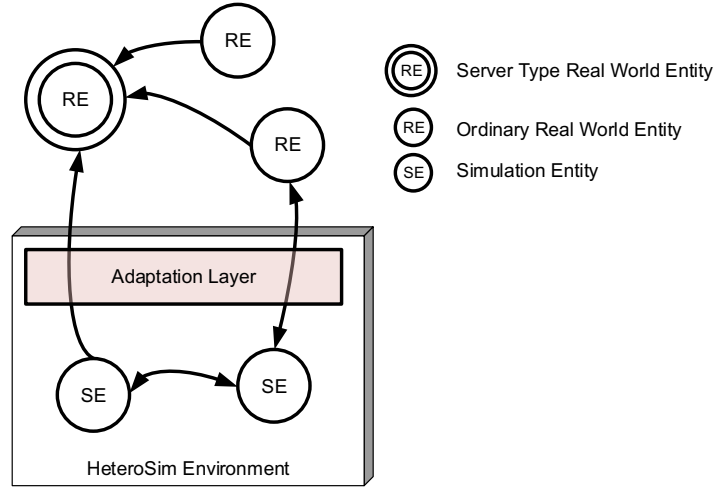
Şekil 5.10 : HeteroSim AWT metrik grafiği.

Bu bölümün devamında, HeteroSim mimarisi anlatılacaktır. Daha sonra evrensel YBH simülasyonu ile Police PBM'in nasıl bir arada çalıştırılabileceği anlatılacaktır.

5.3.1 HeteroSim mimarisi

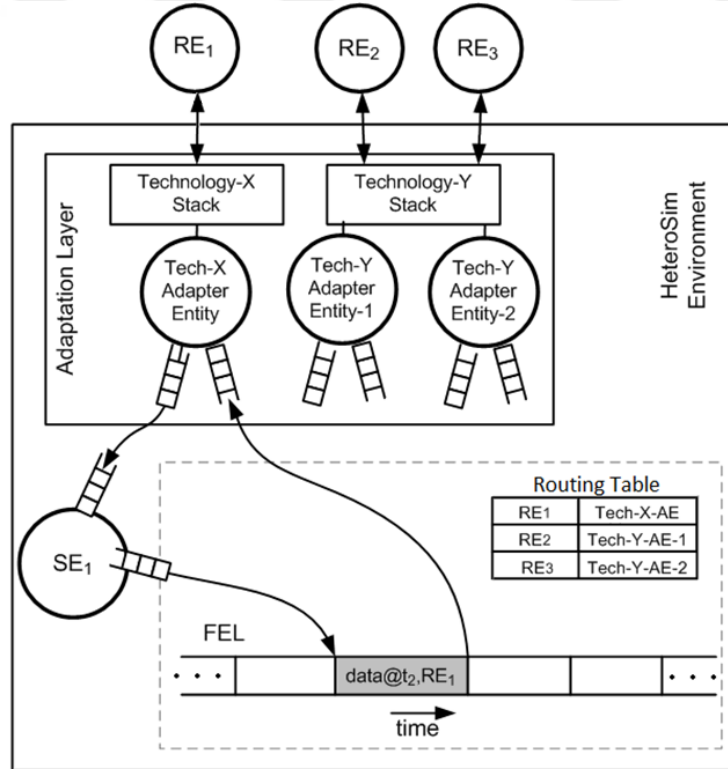
HeteroSim[9], SimJava sayesinde her bir SE'yi (Simulation Entity), bağımsız çalışan Java tabanlı süreçler olarak modeller. SE'ler arasında olay-tabanlı mesajlaşma yapılır. Oluşma zamanına göre sıralanmış olaylar içeren, *merkezi olay kuyruğu* (FEL: Future Event List) vardır. FEL üzerinde en yakın zamanlı olaya simülasyon saati iletildikten sonra yürütme hakkı bu olaya verilir. Böylece sözde paralel (concurrent) bir yürütme şeması gerçekleşir (Gerçek zamana göre bir anda sadece bir süreç çalışır ama simülasyon zamanına göre ise birçok süreç çalışıyor olabilir). O anki simülasyon zamanı için bütün süreçlere yürütme hakkı verildikten sonra simülasyon saati bir sonrakine iletilir.

HeteroSim bileşenleri Şekil 5.11'de gösterilmiştir. Gerçek Dünya Varlıkları (RE), simülasyon yürürken, simülasyon varlıkları (SE) ile haberleşerek simülasyon sonuçlarını etkileyebilirler ve SE'lerin eylemlerinden etkilenebilirler. RE'lerden, dışarıya servis sunanlara, Sunucu Tipli RE (STRE) denilir. STRE'ler, Uygulama Sunucu, RDBMS, JMS Sunucu, WebSunucu, SNMP Etmeni vb. varlıklar olabilir. SE ve RE'ler arasındaki iletişim ya SE ve RE'lerin STRE'lere bağlanması ile ya da RE'lerin simülasyondaki SE'lerle doğrudan etkileşimi ile başlayabilir.



Şekil 5.11 : HeteroSim’de kullanılan kavramlar ve ilişkileri.

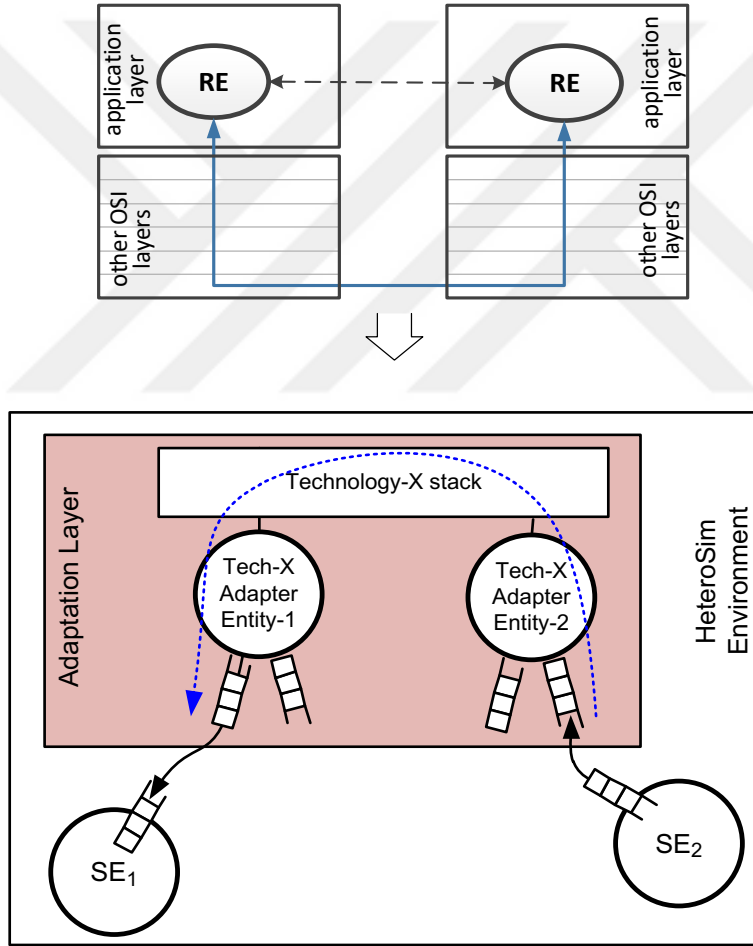
RE’lerin, SE’ler gibi simülasyonun parçası olabilmesi sağlanmıştır. Simülasyon ve gerçek dünya varlıkları, *Uyarlama Katmanı (AL)* adlı bileşen üzerinden haberleşirler. AL, varlıklar arasında hem *multicast* hem broadcast hem de *unicast* iletişimi destekler. AL, SE ve RE’ler arasında, RE’lerin desteklediği çeşitli teknolojilere özel iletişim yolu sağlar (Şekil 5.12). AL bypass edilerek yapılan erişimler, (ki bu sadece SE→RE yönünde olabilir) legal bir simülasyon adımı olarak işlem görmez. Çünkü ne FEL’de bu işleme ait bir olay oluşur ne de olay geçmişine kaydedilir.



Şekil 5.12 : SE ve RE’ler arasında AL üzerinden etkileşim.

Bir iletişim yolu elde etmek için, simülasyonun konfigürasyonuna göre, her bir tipteki RE için bir *Vekil* (Adapter Entity) otomatik olarak oluşur. Vekiller, SE-RE arası trafiği saydam olarak dönüştürüp iletmekten sorumludur. Vekil, RE'ler ile haberleşmek için, RPC, Web Servisi, SMTP, Rest, RMIOP vb. teknolojilerden birisini kullanırken diğer taraftan SE'lerle kendisi de bir SE olarak haberleşir.

AL, simülasyonun doğruluğunu da artırabilir. Bunu, SE'lerin simüle ettiği varlıkların, gerçek dünyada aralarında iletişim için kullandığı teknolojinin aynısını kullanarak sağlar. Böylece daha gerçekçi simülasyonlar düzenlenebilir. İletişim mekanizmalarını simülasyona katmak için fazladan simülasyon kodu yazmaya da gerek yoktur. Şekil 5.13'de iki RE arası iletişimin AL üzerinden simülasyonu gösterilmiştir.



Şekil 5.13 : Teknoloji-X kullanarak haberleşen SE'ler.

Bu örnek senaryoda, SE₂, SE₁'e mesaj göndermek için kullanılacak yol tercihini "*AL Üzerinden*" olarak yaparsa, mesajlar Technology-X iletişim mekanizması kullanılarak iletilecektir. HeteroSim, her ne kadar ağ trafiğini simüle etmese de AL içinde ağ protokolünün kendisini simülasyona dahil ederek doğruluğu artırabilir.

AL mekanizması ile elde edilen diğer bir fırsat ise, farklı bilgisayarlarda veya sanal makinalarda çalışan farklı simülasyonlara ait varlıkların da birbiri ile etkileşebilmesine olanak sağlamasıdır. Bu sayede *dağıtık simülasyon oturumları* düzenlenebilir.

5.3.2 Simülasyon saati sorunu

Simülasyonların hızı, gerçek dünya zamanına göre farklı ayarlanabilir. Zaman kullanımı açısından iki tip simülasyon yürütme modeli vardır [157, 161]:

- Mantıksal Zamanda (Dual Clock): Zamanın ilerlemesi, uygulamanın ilerlemesine bağlıdır. Uygulama saati (simülasyon saati), o anki simülasyon anına ait tüm kodlar yürütülmedikçe bir sonraki ayrık zaman noktasına iletilemez. HeteroSim ve SimJava bu şekilde çalışır.
- Gerçek Zamanda: Simülasyon gerçek saat değerinde ilerler. Alt türleri [161]:
 - *Gerçekçi-zamanda yürütme* (actual-time veya Java yürütmesi), uygulamanın ilerlemesi ile zamanın ilerleyişi birbirinden bağımsızdır. Yani, sistem saati, kaç tane komut yürütüldüğünden bağımsız olarak ilerler. Simülasyon programı ise beklenmeyen durumlar (interrupt, işlemci yükleri ve I/O olayları) nedeniyle değişken hızda ilerler.
 - *Gerçek Zamanlı Yürütme* (real-time): Uygulamanın ilerleyişi zamana bağımlıdır. Belli kodlar, garanti edilen sürelerde tamamlanır. Bu tip simülasyonlar, gerçek-zamanlı sistemlere benzerler.

HeteroSim simülasyonu süresince, FEL içinde, hem gerçek dünya (RE) hem de simülasyon dünyası (SE) varlıklarına dağıtılmayı bekleyen olaylar bulunur. Bu sebeple, simülasyon zamanına ek olarak gerçek dünya saatinin de hesaba katılması gerekir. Olayları simülasyon saatine göre işlemek, RE'ler için sorun oluşturabilir. Simülasyon tarafında yaşanacak bir gecikme, RE'lerin işleyişini olumsuz etkileyebilir. Örneğin, tarih/zaman kısıt ifadesi içeren bir politikanın uygulanması, iki farklı zaman uzayı bulunması nedeniyle tutarlı bir şekilde yapılamaz.

Literatürde hem simülasyon zamanına hem de gerçek dünya zamanına uyumu garanti edebilen bir sistem yoktur. Simülasyonun bir sonraki durumunu hesaplamak için gereken süre, eğer bir sonraki adımın oluşması için kalan süreden daha uzunsa (sapma olacaktır) önlem almak gerekir. Gerçek zaman ile senkron kalabilmek için örneğin, yürütme hızını, gerçek zamana oranla artırmak (örneğin olay listesine, bazı özel

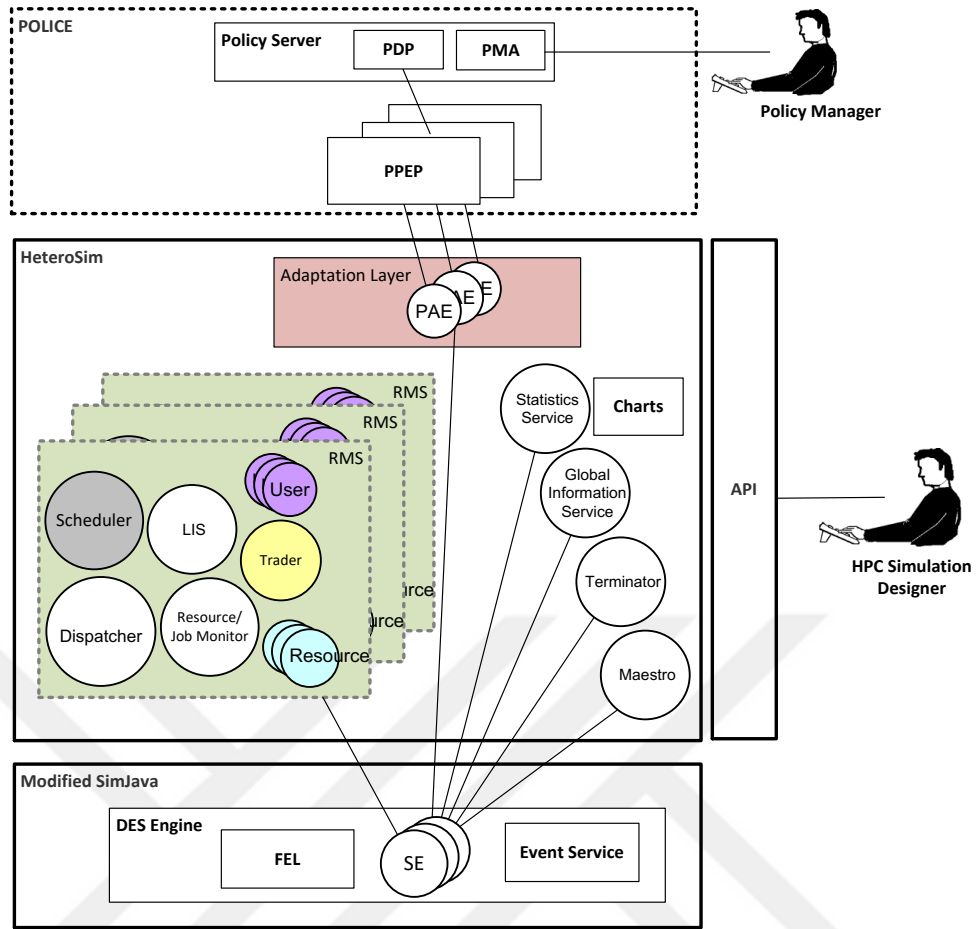
olaylar enjekte etmek veya birim adım süresini değiştirmek), sonraki adım hesaplamasının kalitesini düşürmek veya tamamen ertelemek, gecikmeyi ihmal etmek ve daha sonraki fırsatlarda hızlı çalıştırarak güncel durumu yakalamaya çalışmak gibi alternatifler kullanılabilir. Ancak heterojen varlıklar içeren HeteroSim simülasyonlarında, simülasyon saati ve gerçek dünya saati uyumu, çözülmesi gereken açık bir konudur. HeteroSim’de, çözüme yardımcı olmak için, RE’ler ile ilgili oluşan olaylar, en yüksek öncelikli işlenirler. Sistemde ayrıca, Vekil Varlıklar için, iki yönlü mesaj zaman aşımı (timeout) değerleri tanımlanabilir. Literatürdeki [161], DTS simülasyonu oluşturmak için, JiST (Java in Simülasyon Time) önerilmiştir. JiST, simülasyon saati semantiğini Java dili içine gömerek performans artışı ve saydamlık sağlamaya çalışır. JiST simülasyonun kodunda, gerçek zamana ilişkin fonksiyon çağrıları da vardır. JiST yazılımı, simülasyon programının standart Java-byte-code’larını modifiye edip içine simülasyon yürütme semantiğini gömerek performans artışı elde eder. Bu yöntem, dual zaman problemini çözmemize katkı sağlayabilir.

5.3.3 HeteroSim gerçekleştirilmesi

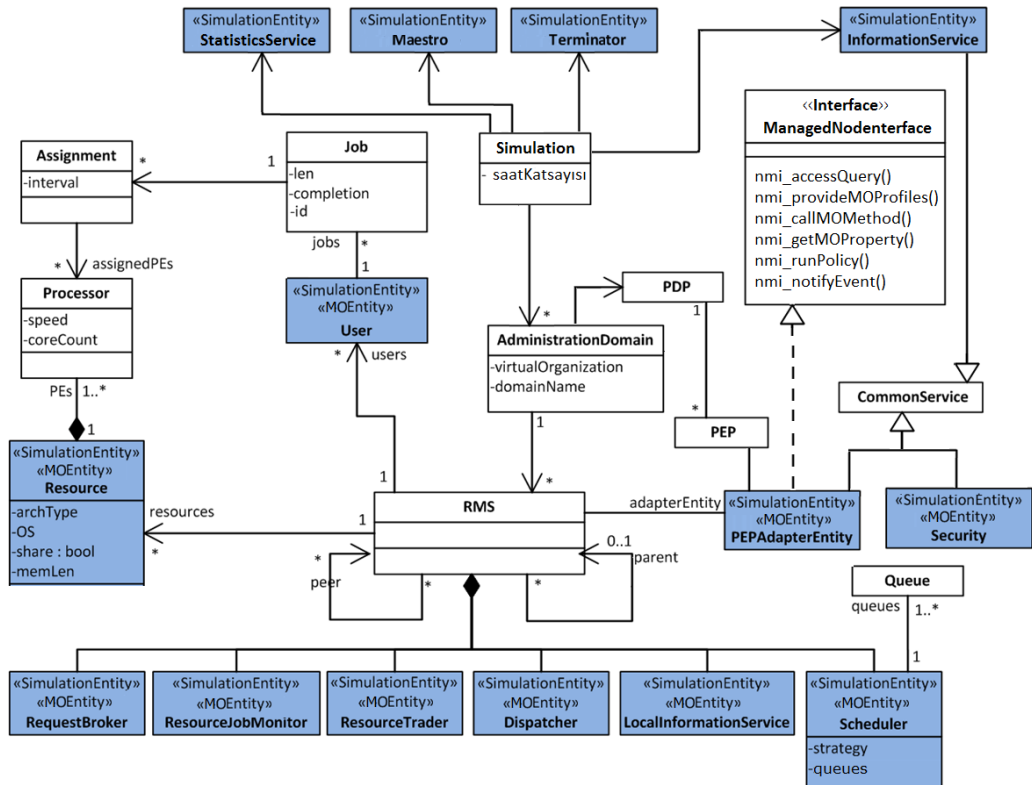
HeteroSim mimarisi Şekil 5.14’de gösterilmiştir. HeteroSim simülasyonlarında YBH modelleri, API çağrıları ile oluşturulabilir veya doğrudan Java kodu olarak yazılabilir. Simülasyonların PBM özellikli olması istenirse en üstteki POLICE katmanı da simülasyonda çalıştırılır. Simülasyon varlıkları otomatik olarak oluşur ve mesajlaşarak çalışır. Bu varlıklar, JVM’de birer Thread olarak çalıştırılırlar. HeteroSim simülasyon ortamı, Şekil 5.15’deki bilgi modeli ile gerçekleştirilmiştir. Modelde, üç tip varlık vardır:

- SimulationEntity: HeteroSim simülasyonundaki varlıklardır.
- MOEntity: Bir PEP’in gözetiminde politikalarla yönetilebilen varlıklardır. Bu tür varlıkların sayısının ileride artması (Job, Queue, Simulation ve hatta İşlemci de içermesi), daha detaylı politika yazılabilmesini sağlayacaktır.
- Diğer varlıklar

HeteroSim, GridSim’den daha detaylı kaynak modellemesi sunabilir. Her bir bilgisayar, bağımsız bir simülasyon varlığına dönüştürülebildiği için İş sıralayıcı, kaynakları bilgisayarlar çözünürlüğünde adresleyebilir (Şekil 5.16). GridSim’de ise ancak bir grup bilgisayar tek bir kaynaktır ve tek bir iş sıralayıcı ile yönetilir. GridSim çözünürlüğünü artırmak için sistem birer bilgisayarlık kaynaklara bölünmelidir ki bu durumda birden fazla iş sıralayıcı da kullanmak gerekir.

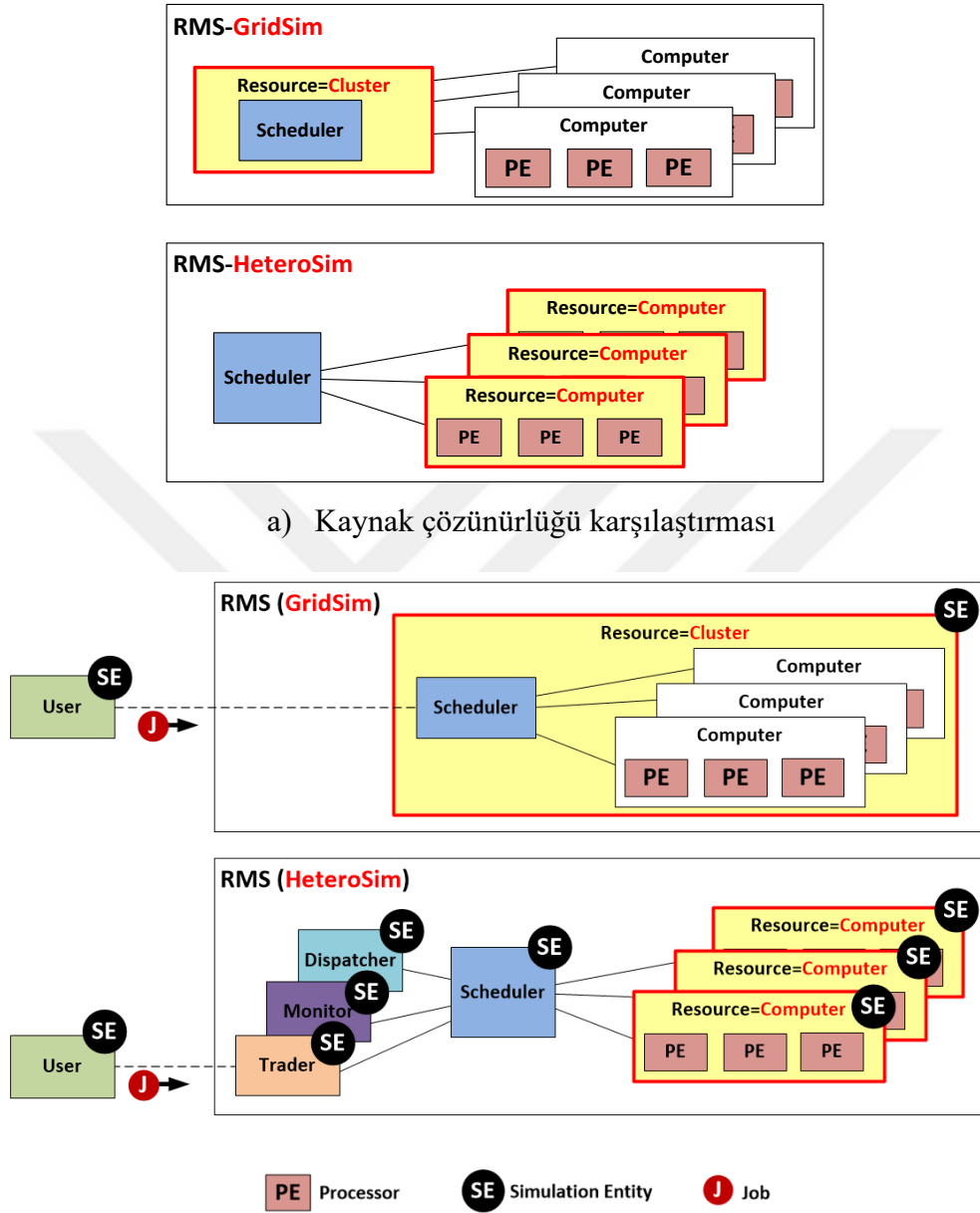


Şekil 5.14 : HeteroSim sistemi blok diyagramı.



Şekil 5.15 : HeteroSim gerçekteşmesine ait bilgi modeli.

HeteroSim’de Kullanıcılar, iş yüklerini, gerçek YBH çalışmasına daha uygun olarak önce Trader’a gönderir. GridSim’de ise doğrudan Scheduler’a gönderildiği için ara bileşenler modellenemez.

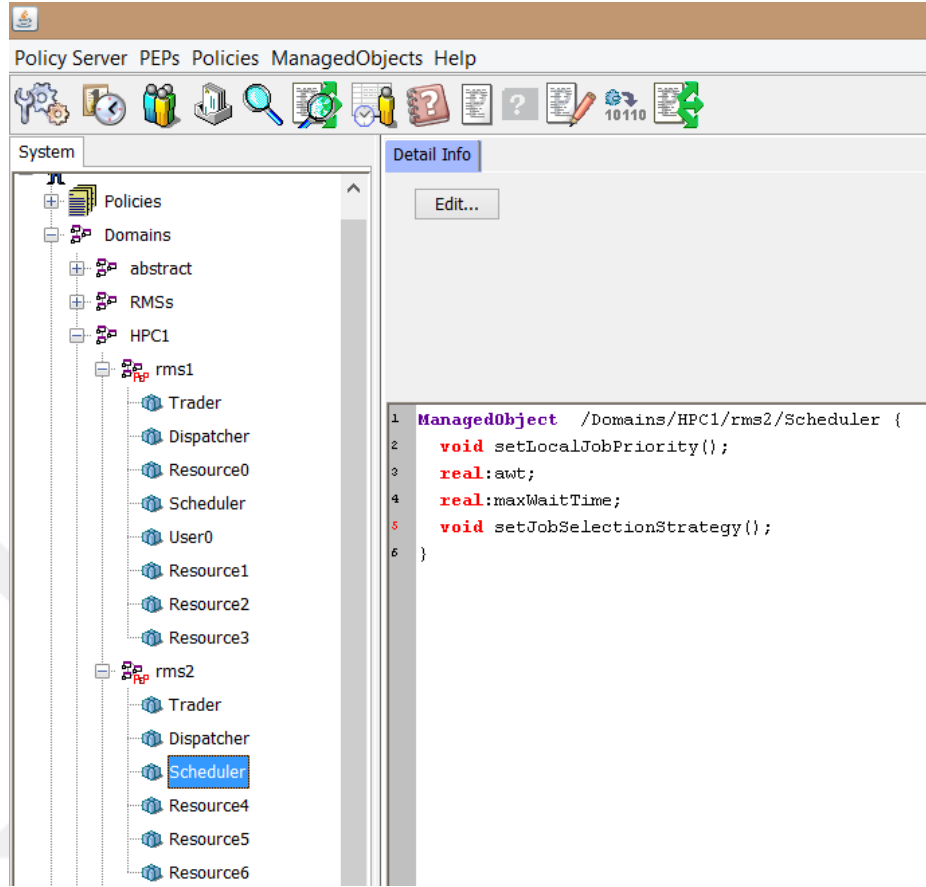


b) Kaynakların SE olarak modellenenebilirliği

Şekil 5.16 : HeteroSim’in yüksek çözünürlüklü kaynak modellemesi.

StatisticsService, metrikleri hesaplayıp simülasyon sonunda gösterir. Metrikler simülasyon sürerken bellekte tutulup hesaplandığı için çok elemanlı simülasyonlarda yavaşlamaya sebep olabilir. RMS metrikleri hesabında, sadece değişiklik oluşan RMS’e ait hesaplamalar yapılarak bu risk azaltılır. İstatistikler, politikaların koşul bölümünde de kullanılabilir. *Scheduler*, *User*, *Resource* ve *YBH* bazında istatistikler

tutulur. Job nesnelerine sistem genelinde tekil kimlikler verilir. Şekil 5.17’de, YBH bileşenlerinin, POLICE politikalarının içinde kullanılabilirliği gösterilmiştir.



Şekil 5.17 : YBH nesnelerinin Managed Object olarak tanımlanması.

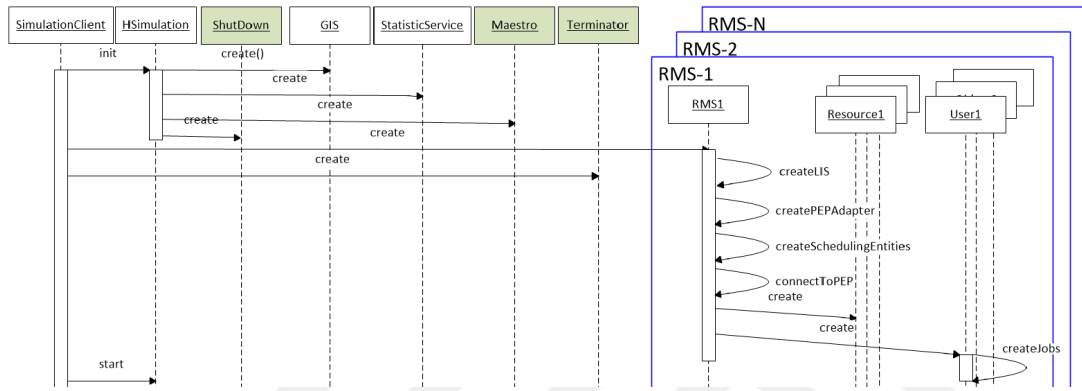
Politikalarla yönetilebilir kılmak için RMS bir PEP’e atanır. PEP-RMS etkileşimi, yine bu RMS’e tahsis edilmiş olan *PEPAdapterEntity* isimli bir *AdaptationLayer* varlığı üzerinden sağlanır. POLICE-HeteroSim entegrasyonunu sağlamak için HeteroSim gerçekleştirilmesinde yapılanlar aşağıdaki listede verilmiştir:

- Simülasyona eklenen konfigürasyon parametresi sayesinde, simülasyonların PEP ile veya PEP’siz modlarda çalışabilmesi sağlanmıştır.
- PEP entegrasyonu için *PEPAdapterEntity* isimli AL sınıfı yazılmıştır.
- Simülasyonun NMI (Node Management Interface) arayüzü üzerinden PEP ile etkileşebilmesi için, simülasyon tarafındaki MO (Managed Object) nesneleri ve arayüzü gerçekleştirilmiştir.
- PBM kontrolünde iş sıralaması yapmak için *HeteroSimPolicyScheduler* isimli iş sıralama sınıfı yazılmıştır. Bu modül, işleri sıralarken PEP’e yaptığı

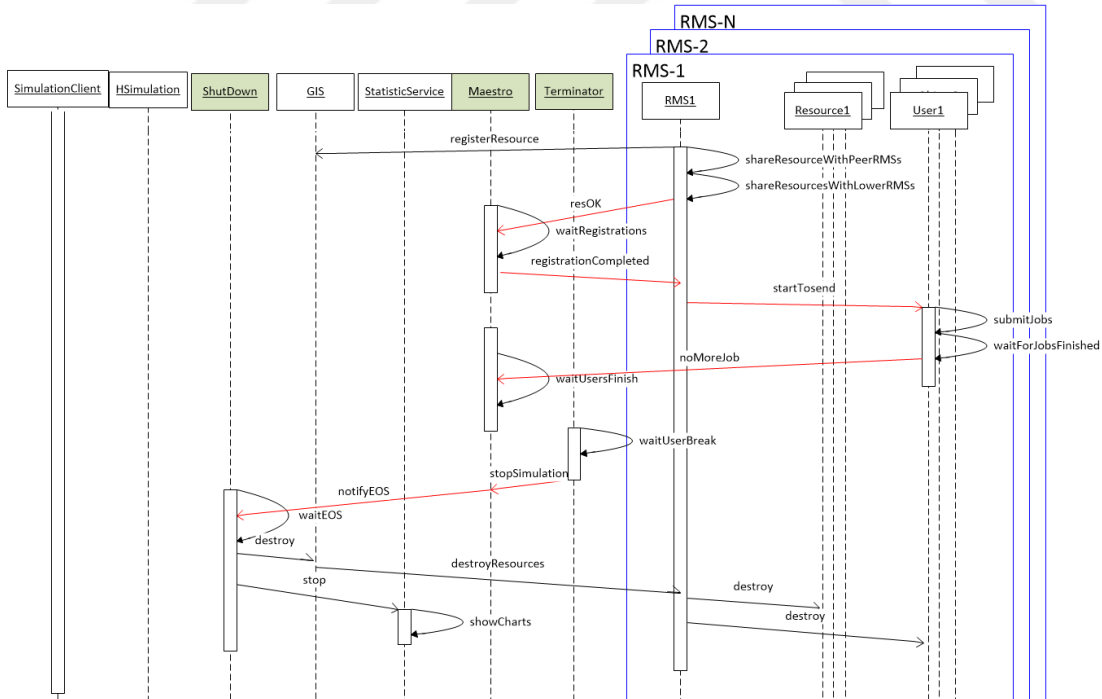
sorguların sonuçlarına göre davranır. Ayrıca, MUSTDO tipli politikaları işleyen PEP, simülasyondaki MO nesneleri üzerinde eylemler başlatabilir.

- YBH'nin kendisi ile ilgili global politikalar uygulayabilmek için PEP₀ isimli özel bir PEP kullanılır.
- Politikalara uyumluluk amacıyla PEP sorguları için gönderilen mesajların, Simülasyon saatini ilerletmemesi sağlanmıştır.

Tasarlanan simülasyonlar çalışırken varlıkların etkileşimi, aşağıdaki UML Sequence diyagramları ile gösterilmiştir (Şekil 5.18, Şekil 5.19). Bu diyagramlarda,



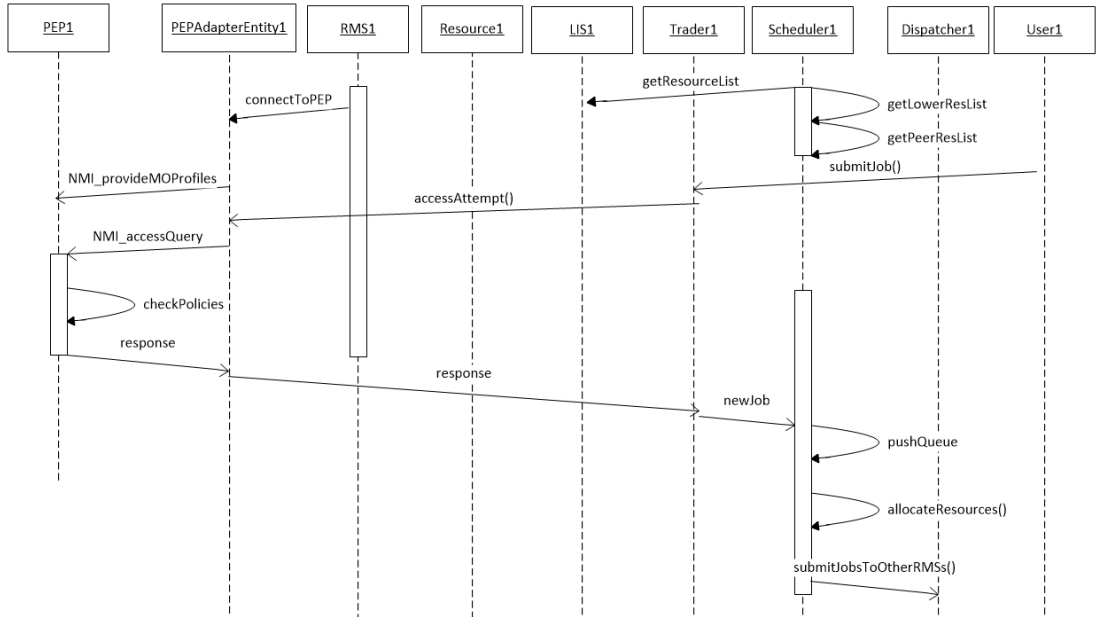
Şekil 5.18 : Simülasyonun başlaması akış diyagramı.



Şekil 5.19 : Simülasyon yürütme ve sonlanma aşaması (Sequence diyagramı).

simülasyonun başlama ve bitiş aşamalarında, simülasyon varlıklarının senkronizasyonu için eklediğimiz (GridSim'de olmayan), Maestro isimli simülasyon

varlığının, diğer varlıklarla etkileşimi de gösterilmiştir. Simülasyon başlarken Simülasyonun ortak varlıkları ve simüle edilecek YBH bileşenleri (RMS, Kullanıcıları ve Kaynakları) oluşturulur. Daha sonra simülasyon başlar (Şekillerde RMS içindeki, Trader, Scheduler, vb. detay varlıklar, basitlik için gizlenmiştir). Maestro, simülasyon başlarken, bütün kaynakların oluşturulduğundan, LIS/GIS'e kaydettirildiğinden emin olur. RMS bileşenleri, kayıt işlemleri tamamlanınca Maestro'ya bildirirler. Maestro, bütün kayıt işlemleri tamamlanınca, sisteme iş gönderecek varlıklara (Kullanıcı, WLTraceLoader vb. varlıklar), iş almaya hazır olduğunu bildirir. İş gönderilip yürütmesi de tamamlanınca, göndericiler Maestro'yu uyarırlar. Maestro, bu uyarıyı alınca simülasyon varlıklarının yok edilmesi sürecisini başlatır. Terminator ise simülasyonun kullanıcı istediğinde kontrollü sonlandırılmasına olanak sağlar. Simülasyonun yürütülme örneği, Şekil 5.20'de gösterilmiştir. Bu örnekte, PBM ile entegre bir yürütme kullanılmıştır. Kullanıcılar, işleri GridSim'dekinin aksine, doğrudan iş sıralayıcıya değil, *Trader* isimli aracıya gönderirler ve pazarlık aşamasını

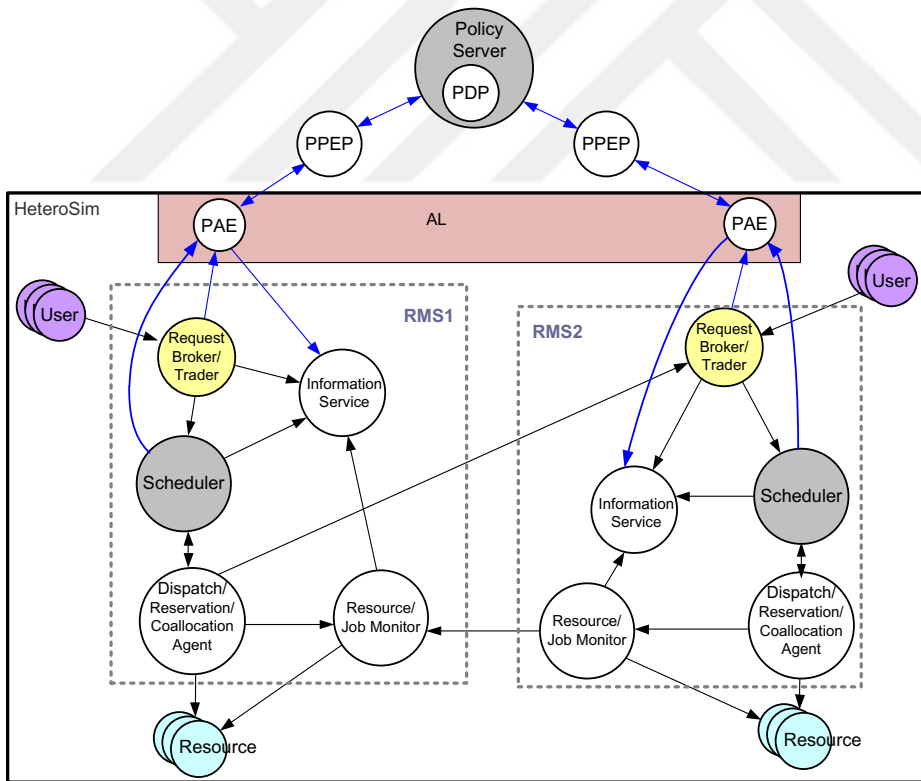


Şekil 5.20 : Politika tabanlı görev yürütme örnek senaryosu.

modelleme imkanı elde edilir. Gelen işler için öncelikle *PEPAdapterEntity* üzerinden PEP'e sorgu gönderilerek *User1*'e erişim izni veren bir politika bulunup bulunmadığı sorgulanır. Cevap olumlu ise görev, *RMS1* iş kuyruğuna girer. Eğer *RMS1*'in iş sıralayıcısı bu görevi başka bir RMS'in (*RMS1*'in kullanımına açtığı) bir kaynağına atarsa, görev *Dispatcher* üzerinden ilgili RMS'in *Trader* bileşenine gönderilir. Her RMS, diğer RMS'ler ile paylaştığı kaynak listesini, kendi LIS bileşenine kaydettirir. LIS ise bu bilgileri, eş ve alt hiyerarşisindeki RMS'lere ait LIS'ler ile paylaşır.

5.3.4 HeteroSim'in PBM simülasyonlarında kullanılması

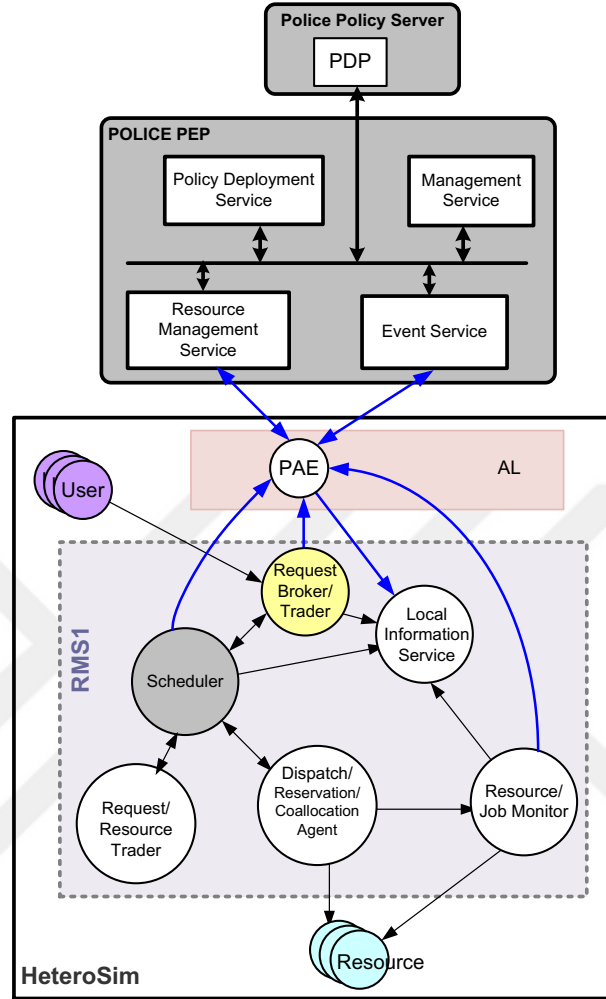
YBH sistemlerinin büyüklüğü ve yetenekleri ile orantılı olarak, yönetilecek parametre sayısı ve yönetim yükü artar. Bu ihtiyaçları karşılayabilmek için PBM'in uygun bir alternatif olduğu, Bölüm 4.1'de irdelenmiş ve Bölüm 4.2'de örneklendirilmiştir. Geliştirilmiş araçlarda politika ve strateji kavramları, bir PBM sisteminde olmaları gereken anlamlardan farklı olarak kullanılmıştır ve bu araçlar PBM karakteristiklerini tam anlamıyla göstermezler. YBH sistemlerinin PBM ile uyumlu yönetiminin mümkün olduğunu PBM'in, YBH yönetiminde kullanılabilirliğini göstermek için Police PBM aracı, HeteroSim aracına entegre edilerek simülasyonlarda politikalarla çalışmak hedeflenmiştir. HeteroSim içinde, bilinen YBH sistemlerini modelleyebilmek için gerekli simülasyon varlıkları (SE), Bölüm 4.3'de tasarlanan modelimize uygun olarak gerçekleştirilmiştir. Geliştirilen simülasyonların Police PBM aracı ile yönetimi için, HeteroSim'in gerçek dünya yazılımları ile etkileşebilme yeteneği kullanılmıştır (Şekil 5.21).



Şekil 5.21 : Police'in HeteroSim YBH simülasyonu ile entegrasyonu.

Politika yürütmek için, SE'ler ile Police PBM bileşenleri, HeteroSim'in Uyarılma Katmanı (AL) üzerinden birbirine bağlanır. Şekil 5.21'deki senaryoda, iki RMS içeren YBH simülasyonunda Police PEP kullanımı gösterilmiştir. Police bileşenleri,

simülasyondaki gerçek dünya varlıklarını (RE) temsil eder ve YBH bileşenlerini temsil eden simülasyon varlıkları (SE) ile etkileşirler (Şekil 5.22).



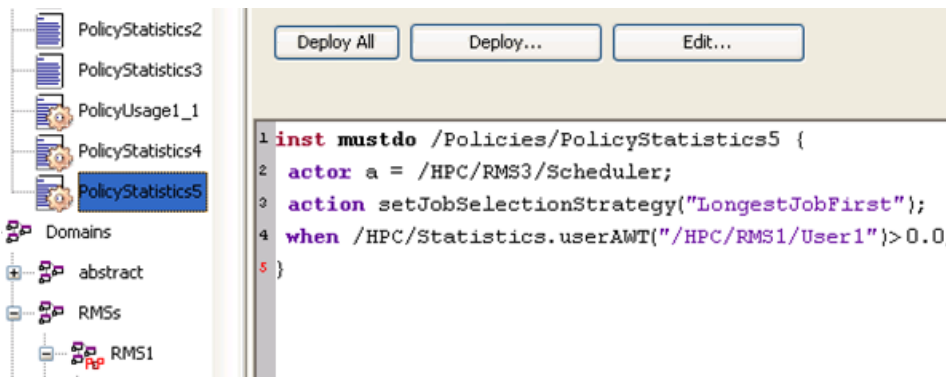
Şekil 5.22 : Police PEP'in, HeteroSim'de bir RMS ile etkileşim bağlantısı.

Bu senaryoda, AL içindeki Vekil rolünü, RMS için oluşturulan ve ilgili PEP ile haberleşen PAE (*PEP Adapter Entity*) oynar. PEP tarafından yönetilen nesneler (managed objects), RMS'in alt bileşenleri, kullanıcıları ve kaynaklarıdır. Simülasyon başlarken PAE, bu yönetilen nesneleri PPEP'e kayıt ettirir. Buradaki Aktör nesneler, kullanıcılar ve VO'lardır (Virtual Organization). Hedef nesneler ise kaynaklardır. Police politikalarının içinde, *İş*, *İş Kuyruğu*, *Kullanıcılar*, *Kaynaklar*, *İş Sıralayıcılar*, *Broker*, *Trader* bileşenleri, aktör/nesne olarak kullanılabilir. Police politika dili, *YBH*, *Statistics*, *RMS* ve *Scheduler* anahtar kelimelerini de içerecek şekilde genişletilmiştir. Ayrıca, *İş* sıralayıcıya kuyruklardan iş seçme stratejileri eklenmiştir. Bu sayede, iş sıralayıcıların çeşitli davranış kalıpları, politikalarla ayarlanabilir (Örnek, *LONGEST_FIRST*, *LOCAL_FIRST*, *EQUAL*, *REMOTE_FIRST* ve

LEAST_REMAINING_FIRST). Politikaların içinde kullanılabilecek örnek POLICE politika ifadeleri aşağıda listelenmiştir:

- rms1.scheduler.awt(user)
- rms1.scheduler.awt()
- rms1.hpc.awt()
- getResponsiblePEPofMO(rms1.hbc)
- findQueueName("rms1/hpc")
- rms1.user.awt(user3)
- hpc.rms2.user1.aet
- rms1.scheduler.awt()
- rms1.scheduler.hpcAWT()
- rms1.scheduler.userAWT(userX)
- hpc.awt
- hpc.loadRatio
- rms1.scheduler.setJobSelectionStrategy("LocalFirst")

Politikalarda iş yüklerinin kaynaklara atanması, iş yükü sahibi kullanıcının, hedef kaynak üzerinde bir erişim isteğinde bulunması olarak modellenir (*hedefKaynak.use()* adlı bir metoda erişim şeklinde modellenir). Örneğin, aşağıdaki POLICE politikası, RMS3'e ait iş sıralayıcı bileşenince kullanılan iş seçim stratejisini, *User1* için ortalama iş bekleme süresi (AWT) sıfırdan büyük olduğunda değiştirir. Öyle ki bekleyenler işlerden en uzununu daha öncelikli olur.



Şekil 5.23 : Örnek HPC yönetim politikası.

RMS'ler, birlikte çalışan bileşenleri ile sonuç üretirler. Dolayısıyla, sadece iş sıralayıcının değil bu RMS bileşenlerinin hepsinin bir bütün olarak yönetilebilmesi gerekir. Aksi halde, sistemde tutarsızlıklar oluşabilir. Örneğin, sadece erişim

denetiminin politikalarla yönetildiği ama iş sıralayıcının yönetilmediği durumda, işler, erişim yetkileri olmayan kaynaklara atanabilirler.

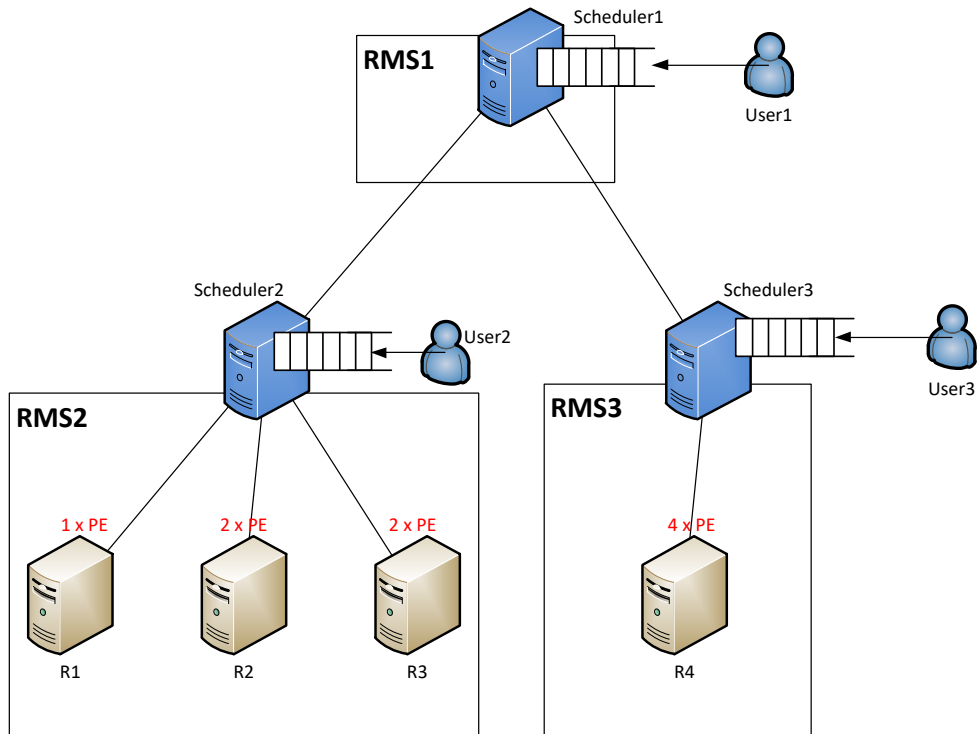
RMS modelimizde, farklı yönetimsel uzaylara ait olsa da PBM ile yönetilebilir hiyerarşik RMS'lerin topolojileri oluşturulabilir. Her yönetim uzayına ait PDP, kendi RMS'leri için kendi politikalarını tanımlayabilir. Uzaylar arası kaynak paylaşımı, politika pazarlığı ve SLA'lar vasıtasıyla yapılabilir hale gelir. Bu pazarlık konuları gelecekte yapılacak işler listemizdedir.

Kaynakların tahsisinde politika kullanımının mümkün olması yanında, Police'in sağladığı E-Code yeteneği sayesinde, işlerin kaynaklara atanmasını kısıtlayabilecek, sisteme özel kısıtların da kotarılması mümkün olur. YBH sistem yöneticileri ve/veya YBH sistem üreticileri, kaynak atama sırasında dikkate alınması gereken semantik kısıtlara karşı düşen E-Code'ları sağlayabilirler.

Simülatör yazılımlarının en önemli kalite karakteristikleri, ölçeklenebilirlik ve doğruluktur. HeteroSim'in bu iki karakteristiği aşağıda ele alınmıştır.

5.3.4.1 HeteroSim doğruluk analizi

HeteroSim'in, tanımlanan modelleri doğru olarak çalıştırdığı ve literatürdeki RMS sistemlerinden beklenen davranışları sergilediği, Şekil 5.24 örnek mimarisi üzerinde



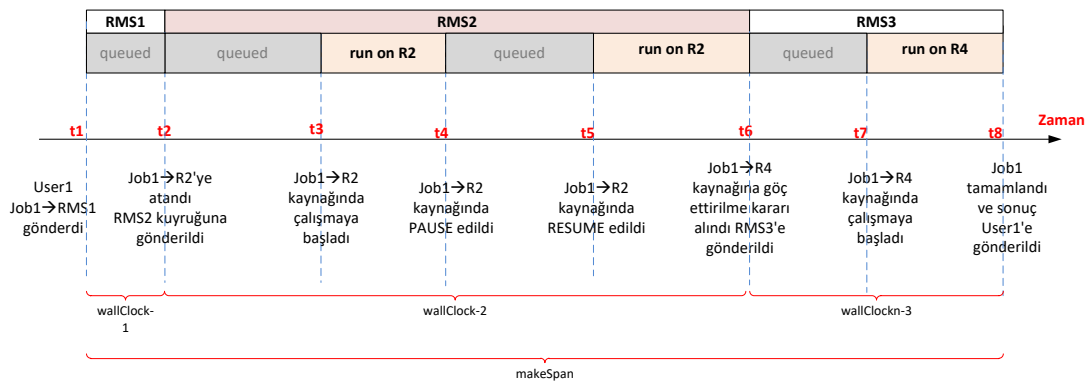
Şekil 5.24 : Doğruluk analizinde kullanılan YBH mimarisi.

bu alt bölümde gösterilmiştir. Bu amaçla bir dizi deney gerçekleştirilmiş ve sonuçları bu bölümde paylaşılmıştır. Deneylerde, Şekil 5.24'deki YBH mimarisi çeşitli yükler altında simüle edilmiştir. Bu mimari, RMS hiyerarşileri arasında taşınan iş yüklerini, yerel ve uzak RMS'e ait iş yüklerinin ele alınışını gösterilebilecek şekilde seçilmiştir.

HeteroSim'de önemli sınıflardan olan *Job (J)* sınıfı içinde başlıca şu veriler bulunur:

- jobLength: çalıştırılacak işin büyüklüğü (birim MI)
- numPE: işi çalıştırmak için gerekli işlemci sayısı
- status: işin güncel durumu (CANCEL/INEXEC/PAUSED/RESUMED/...)
- finishTime: işin tamamlandığı zaman
- firstExecTime: en son çalıştırmaya başlama zamanı
- submitTime: işin, YBH sisteme giriş zamanı
- finishedSoFar: işin herhangi bir simülasyon anındaki biten uzunluğu
- resBilgi[]: işin her bir kaynak üzerindeki çalışma geçmişi
 - submissionTime: işin kaynağa gönderilme zamanı
 - execStartTime: atanan tüm kaynaklarda ortak başlangıç zamanı
 - arrivalTime: işin geliş zamanı
 - wallClockTime: işin, bir kaynağın içinde bulunma süresi (ayrılma zamanı - gelme zamanı)
 - actualCPUTime: İşin, beklemler hariç toplam yürütme süresi.
 - Ri.PE[]: işe tahsis edilen işlemci kimlikleri

Tutulan bu detaylı bilgi sayesinde işlerin yürütmesine ilişkin her türlü metrik hesaplanabilir. Yukarıdaki iş yükü ile ilgili kavramların daha iyi anlaşılabilmesi için bu parametrelerin, yine yukarıdaki şekildeki mimaride cereyan eden örnek bir senaryo sırasında aldığı değerler gösterilmiştir.



Şekil 5.25 : Örnek iş yükü yaşam çevrimi.

Parametrelerin, bu senaryonun önemli anlarındaki değişimi Çizelge 5.2’de açıklanmıştır. Bu örnek İş yükü, 1.500 MI (MegaInstruction) uzunluğunda ve 2 adet PE’de paralel çalışma ihtiyacındadır. R2 işlemcileri 200MHz, R4 işlemcileri ise 400MHz hızında kabul edilmiştir. R2 ve R4, dış RMS’ler ile paylaşılan kaynaktır.

Çizelge 5.2 : Örnek senaryo aşamalarında parametrelerin değişimi.

Zaman	Değiştirilen parametreler	Açıklama
T0	User1 iş yükünü oluşturdu J.len= 1.500 J.numPE= 2	
T1	J.execSubmitTime= t1 J.resBilgi[1].kaynak= R2	RMS1’de yerel kaynak yok. Alt hiyerarşideki RMS2’in paylaştığı R2 kaynağı seçildi
T2	J.resBilgi[1].arrivalTime= t2 J.resBilgi[0].wallClockTime= t3-t1	
T3	J.resBilgi[1].execStartTime= t3	
T4	J.finishedSoFar= (t4-t3).200 J.resBilgi[1].actualCPUtime= t4-t3 J.status= PAUSED	R2’nin kesinti zamanı geldi. İş askıya alındı
T5	J.status= RESUMED	
T6	J.resBilgi[2].kaynak= R4 J.resBilgi[2].submitTime= t6 J.finishedSoFar= [(t6-t5)+(t4-t3)].200 J.resBilgi[1].actualCPUtime= (t4-t3)+(t6-t5) J.resBilgi[1].wallClockTime= t6-t2	
T7	J.resBilgi[2].execStartTime= t7	
T8	J.resBilgi[2].finishTime= t8 J.finishedSoFar= 1500 J.resBilgi[2].actualCPUtime= (t8-t7) J.resBilgi[1].wallClockTime= t8-t6	

Bu bölümdeki deneylerde kullanılan kaynakların özellikleri, Çizelge 5.3’de verilmiştir.

Çizelge 5.3 : Kaynakların özellikleri.

RMS Adı	Hesaplama kaynağı	Özellikler	PE Hızı (MIPS)
RMS1	Broker	Kaynağı yok	
	Resource1	1 x PE	500
RMS2	Resource2	2 x PE	500
	Resource3	2 x PE	500
RMS3	Resource4	4 x PE	1.000

Deneylerde kullanılacak örnek iş yükü listesi ise Çizelge 5.4’de verilmiştir. İşlerden bazıları (Job1, Job7, Job8), çalışmak için birden fazla işlemcinin tahsisini gerektiren paralel iş yükü olarak tanımlanmıştır.

Çizelge 5.4 : İş yüklerinin özellikleri..

Kullanıcı	İş	Gerekli İşlemci sayısı	İş uzunluğu (MI)
User1	Job#1	3	20.000
User2	Job#2	1	30.000
	Job#3	1	5.000
	Job#4	1	50.000
	Job#5	1	35.000
	Job#6	1	3.000
	Job#7	4	45.000
User3	Job#8	3	38.000
	Job#9	1	150.000
	Job#10	1	10.000
	Job#11	1	2.000
	Job#12	1	20.000

Daha sonra örnek kaynak atama geçmişi diyagramları üzerinden bu parametrelerin değerlerinin nasıl değiştiği irdelenmiştir.

Doğruluk analizinde kullanılan metriklerin tanıtımı izleyen alt bölümde yapılmıştır. Daha sonraki alt bölümde ise düzenlenen deneyler ve sonuçları paylaşılmıştır.

Literatürde, YBH çalışmalarında niceliksel olarak değerlendirmelerde bulunmak için pek çok metrik kullanılmıştır [198][199]. Bu metrikler bölümün geri kalan kısmında iki grup altında listelenmiştir. YBH sistemlerinde çok kullanılan metriklerin tanımları aşağıdaki gibi iki grup halinde verilmiştir:

- Literatürde sıkça kullanılan Kullanıcı’nın bakış açısından metrikler: Kullanıcı için sistemin cevap süresi (turnaround) ve bekleme süresi önemlidir. Bekleme süresi, kısa iş yükleri çalıştıran kullanıcılar için daha önemlidir. Bu amaçlı metrikler Çizelge 5.5’de açıklanmıştır.
- Literatürde sıkça kullanılan Sistem yöneticisinin bakış açısından metrikler: Sistem yöneticisinin gözüyle sistemin kendi performansını değerlendirmek üzere kullanılabilecek metrikler kümesi ise Çizelge 5.6’da verilmiştir.

Çizelge 5.5 : Kullanıcı açısından metrikler (N adet iş yükü için).

Metrik	Açıklama
ART (Average Response Time veya Aggregated Response Time) veya ATT (Average Turnaround Time)	Herbir iş yükünün gelişi ile bitişi arası süre ortalaması. $\frac{1}{N} \sum_{j \in Jobs} EndTime_j - SubmitTime_j$ Not: Tüm iş yükleri aynı sunucuda eş zamanlı çalışmak isterse hepsi gecikir ve ATT artar. Eğer 1 sn.de tamamlanabilecek 3 iş yükü kaynak üzerinde aynı anda (concurrent) çalışırsa (adil paylaşım), her iş yükü 3 saniye çalışır ve ATT=3s olur (Tüm işler 3.saniye sonunda tamamlanır). Eğer 3 iş yükü (sequential) çalışırsa, her iş yükü 1 saniye çalışır ve ATT 2sn olur (1.saniye 1.iş, 2.saniye 2.iş, 3.sn sonunda 3. İş biter ve sonuçta ortalama zaman (1+2+3)/3 = 2 saniye). Yani, ATT tek başına sistem performansını ölçmek için yeterli değildir.
Makespan (Total Execution Time)	İlk iş yükünün yürütmeye başlama zamanı ile sonuncunun tamamlanma zamanı arasındaki süre.
Fairness	Adillik. Kullanıcıların paylaştığı kaynak miktarına göre yürüttürdükleri işlerin orantısı.
AWT (Average Wait Time)	$\frac{1}{N} \sum_{j \in Jobs} WaitTime_j \quad WaitTime_j = StartTime_j - SubmitTime_j$
AWWT (Average Weighted Wait Time)	İşin çalışmaya kadar ne kadar beklediği (Kullanıcı açısından iş planlama kalitesini ölçer). $\frac{\sum_{j \in Jobs} SA_j \cdot WaitTime_j}{Total SA}$ $SA(SquashedArea) = \sum_{j \in Jobs} coreUsed_j \cdot runtime_j$
AWTD (Average Wait Time Deviation)	$\frac{1}{N} \sqrt{\sum_{j \in Jobs} (WaitTime_j)^2 - \left(\sum_{j \in Jobs} (WaitTime_j / N) \right)^2}$
AET (Average Execution Time)	Kullanıcıya/Sisteme/RMS'e ait işlerin yürütme sürelerinin ortalaması
Over-deadline	İşin QoS talebinin ne kadar karşılandığını ölçer = $\sum$ işin deadline'ının ne kadar aşıldığı. Kaynak kullanımı, sistem-merkezli metrik olarak da çok kullanılır

Çizelge 5.6 : Sistem açısından metrikler.

Metrik	Açıklama
YBH Efficiency	Kullanımdaki ve boştaki kaynakların oranı $\frac{\sum_{j \in Jobs} (EndTime_j - StartTime_j) \times CPUs_j \times CPUSpeed_j}{(EndTime_{last\_job} - SubmitTime_{first\_job}) \times \sum_{m \in Servers} CPUs_m \times CPUSpeed_m} \times 100\%$ Payda: tüm simülasyon boyunca her iş yükünün kullandığı cpu sayısı ve hızlarının, toplam sistem cpu ve hız çarpımına oranı
System Admission Ratio (Job acceptance rate)	Belli zaman diliminde, başarılı olarak kabul edilen toplam iş yükü çalıştırma isteği / toplam istekler. Kullanıcı istekleri yeterli doğrulukta olmadığı için bu metrik iyi bir ölçü olamaz..
Job Rejection Rate	Adillik. Kullanıcıların paylaştığı kaynak miktarına göre yürüttürdükleri işlerin orantısı.
AWT (Average Wait Time)	Reddedilen iş yüklerinin toplam yüzdesi
Average Computational Resource Utilization (ACRU)	$\frac{\sum_{j \in Jobs_{CR}} Load_j}{Makespan \cdot speed_{CR}}$ Makespan süresince (Δt) atanan CR hızına oranı.
AgRU (Aggregated Resource Utilization)	$AgRU = \frac{\sum ET_i}{\#cpu \cdot \Delta t}$ ET_i kullanıcılarınca tüketilen CPU kaynaklarının toplam CPU kaynaklarına oranı. Kaynak sahibinin kısıtlarının ne kadar iyi uygulandığını gösterir.
ARU (Average Resource Utilization)	Kuyrukta bekleme ve askıda kalma zaman dilimleri hariç, iş yüklerinin kaynakları kullanımını ölçer.
Başarım (Throughput)	Kuyrukta bekleme ve askıda kalma zaman dilimleri hariç, iş yüklerinin kaynakları kullanımını ölçer.
Başka RMS'e göçen iş oranı	İş bitirme hızı
Başka RMS'e göçen iş oranı	Transfer edilen (göç ettirilen) iş yükleri / toplam iş yükleri
AET (Average Execution Time)	Kaynaklarda işlerin yürütme sürelerinin ortalaması

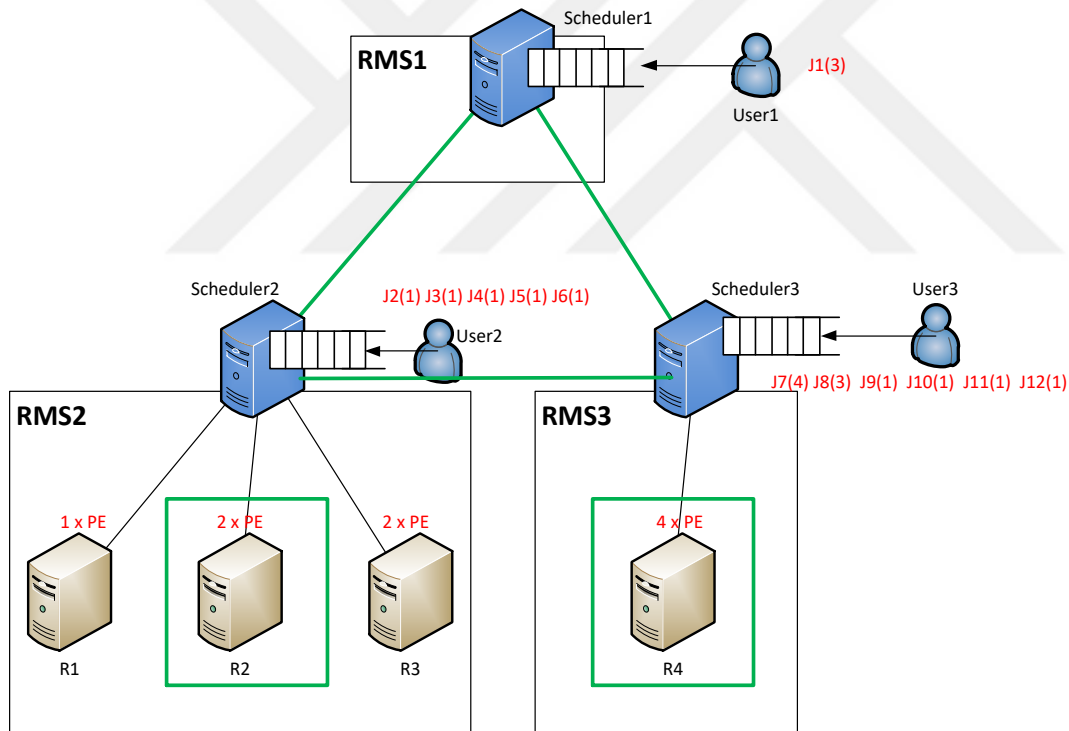
Çizelge 5.6 (devam): Sistem açısından metrikler.

Metrik	Açıklama
Başka RMS'e göçen iş oranı	Transfer edilen (göç ettirilen) iş yükleri / toplam iş yükleri
AWSD (Average Weighted SlowDown)	Slow Down (sistemde harcadığı sürenin gerçek çalışma süresine oranı) $SD = \frac{endTime_j - submitTime_j}{runTime_j} = (waitTime + runTime) / runTime$ $AWSD = \frac{\sum_{j \in Jobs} SA_j \cdot SD_j}{Total\ SA}$
AWRT (Average Weighted Response Time)	İş yükü planlama kalitesini ölçmek için kullanılan ART (İş yüklerinin kaynak tüketimine göre ağırlıklandırılmış hali). Bir işin response time'ı, tükettiği kaynaklarla ağırlıklandırılarak küçük işlerin bu metrik üzerindeki etkisinin büyük işlerden fazla olması engellenir. $AWRT = \frac{\sum_{j \in Jobs} SA_j \cdot (endTime_j - submitTime_j)}{Total\ SA}$ $SA_j = reqResources_j \cdot (endTime_j - startTime_j)$
ASF (Average Starvation Factor)	Boşta ama bir kullanıcıya verilmeyen kaynakların, bir ET _i kullanıcısı tarafından kullanılan CPU sayısına oranı. $ASF = \frac{\sum MIN(ST_i, RT_i)}{\sum ET_i}$ ST _i : bir kullanıcı tarafından istenen ama verilmeyen CPU sayısı, RT _i : toplam kaynak
AJC (Aggregated Job Completion)	Tamamlanan iş sayısının gönderilmiş iş sayısına oranı
Lokal olarak hizmet verilen iş yükü sayısı	Kullanıcı tarafından gönderildiği RMS düğümünde hizmet alan işlerin sayısı (veya toplam iş sayısına oranı)
Uzak bir RMS'e gönderilerek hizmet verilen iş yükü sayısı	İlk gönderildiği RMS düğümünden farklı bir RMS düğümüne iletilerek hizmet alan işlerin sayısı (veya toplam iş sayısına oranı)

HeteroSim simülayonlarında bu metriklerden, literatürde en bilinenler kullanılmıştır. Metriklerin hesabı için, iş yürütme detayları sistem tarafından depolanır. Çizelge 5.1’de verilen, HeteroSim içinde gerçekleşen bu metriklere HeteroSim’in esnek metrik toplama mekanizması sayesinde yenileri kolayca eklenebilmektedir. Yapılan deneyler bundan sonra sırasıyla paylaşılmaktadır.

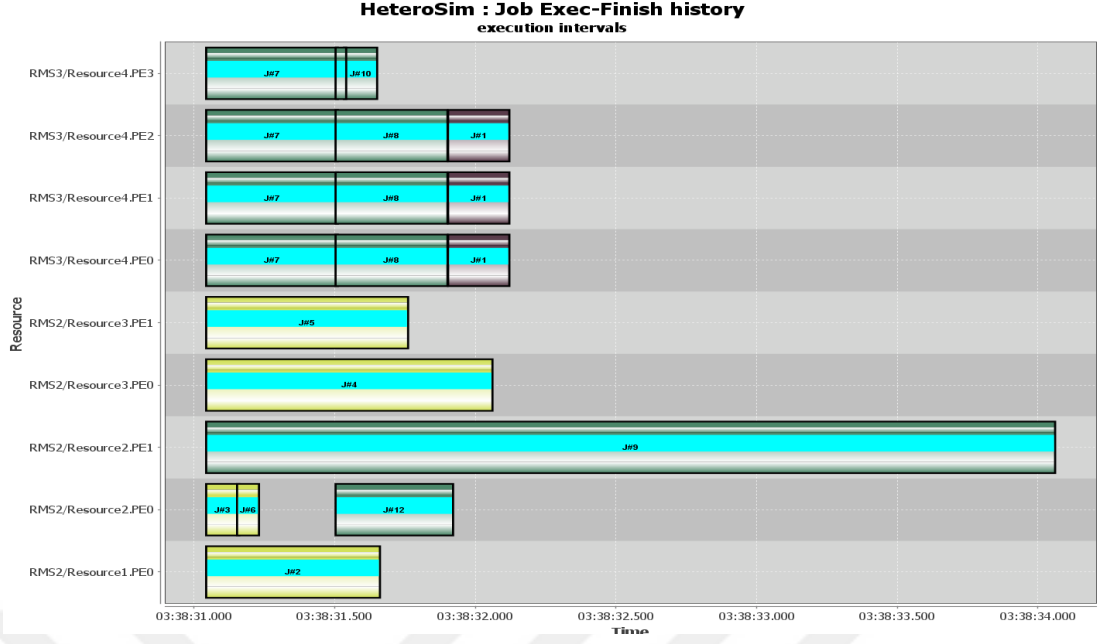
DENEY-1

Bu deneyde, R2 ve R4, dış RMS’lere de paylaşılmıştır ve RMS2 ile RMS3, birbirlerine peer RMS olarak tanımlanmıştır. Şekil 5.26’daki mimaride Yeşil çerçeve içine alınmış kaynaklar, diğer RMS kullanıcılarının kullanımına açılmış olanlardır. Bu sayede, simülasyon sırasında, RMS3’ün kullanıcısı olan User3’e ait Job#09 ve Job#12, simülasyon sırasında R2’de çalışmıştır.



Şekil 5.26 : Deney1 mimarisi.

Kullanıcılar, testin başında işlerini aynı anda sisteme gönderirler. Sistemde ikiden fazla işlemcisi olan R4 kaynağına, Job#7, Job#8 ve Job#1 bu kaynağı sırayla atanmıştır. Kaynaklar dolu olduğu için Job#06, Job#01, Job#10, Job#11 ve Job#12, kuyruklarda bir süre beklemek zorunda kalmıştır. Simülasyon süresince işlerin kaynaklarda çalışma zamanları Şekil 5.27’de gösterilmiştir.



Şekil 5.27 : Deney1 sonucu iş-kaynak atama grafiği.

Simülasyon sırasında olanları iş yükü bazında incelersek:

- Job#1, atanabileceği tek kaynak olan R4 kaynağının boşalmasını beklemiştir.
- Job#6'nın atanabileceği kaynaklar R1, R2, R3 ve R4'tür. Hepsi dolu olduğu için beklemiştir. İlk boşalan kaynak olan R2'ye atanmıştır.
- Job#10, Job#11, Job#12'in atanabilecekleri kaynaklar sadece R2 ve R4'tür. R4 doludur. Bu işler R2'ye daha erken atanabilecek iken atanmamıştır. Bu ise RMS3.Scheduler'da bekleyen bu işler, RMS2.Scheduler'daki iş yükü değişikliğinden geç haberdar edilmiştir. Komşu RMS'deki değişikliklerin diğer RMS'lere bildirilmesi için gereken merkezi izleme mekanizmasının eksikliği, HeteroSim'in iyileştirilmesi gereken bir noktasıdır.

GridStatistics'in, deney sonunda verdiği global metrik değerleri aşağıda verilmiştir:

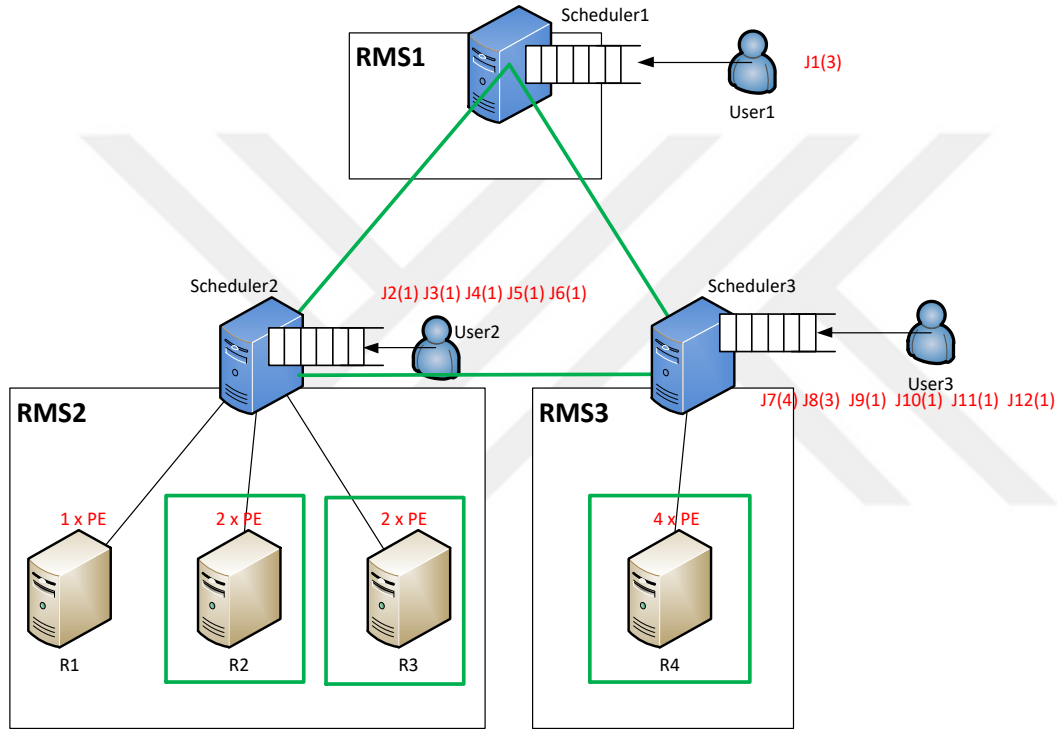
- AWT: 236.83333333333334
- MakeSpan: 301.0 (en uzun iş Job#09 için)
- Mean Job Length: 34.000.0 (408.000 /12 adet)

Bütün işler tek bir kaynakta ve bir çalıştırmada (pause edilmeden) çalıştırıldığı için Ri.wallClock = makeSpan çıkmıştır. Bütün simülasyonun için makeSpan değeri, en uzun iş olan Job#09'un makeSpan değerine eşit çıkmıştır. İşlerin çalışma süreleri incelendiğinde, çıkması gereken çalışma süreleri, yukarıdaki tablodaki cpuTime kolonundan birer saniye daha kısa olarak görülür. Bunun sebebi, HeteroSim

checkPoint timeout'unun 1 saniye olarak konfigüre edilmiş olmasıdır. İş boyları büyüdükçe bu 1 saniye, ihmal edilebilir. Bununla birlikte, HeteroSim'de bu parametre küçültülerek daha küçük sapma değerleri de elde edilebilir.

DENEY-2

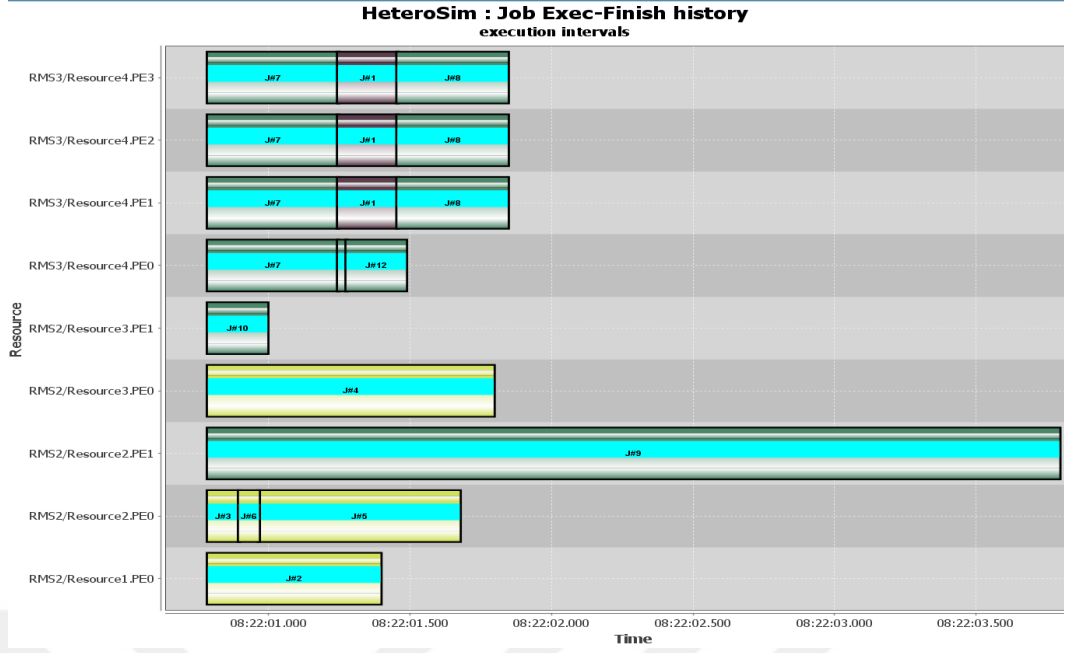
Bu deneyde, R2 ve R4, dış RMS'lerin paylaşımına da açılmıştır. Ayrıca RMS2 ile RMS3, birbirlerine peer RMS olarak tanımlanmıştır. Deney-1'en farkı, RS2'deki R3 kaynağının da dış paylaşımına açılmış olmasıdır (Şekil 5.28). Bütün kullanıcılar, testin başlaması ile birlikte, listelerindeki işleri aynı anda sisteme gönderirler.



Şekil 5.28 : Deney2 mimarisi.

Simülasyon süresince işlerin kaynaklarda çalışma zamanları Şekil 5.29'da gösterilmiştir. Metrik grafikleri, önceki deneyden çok önemli değişiklikler içermediği için bu deney sonucunda paylaşılmamıştır. GridStatistics servisinin, koşturma sonunda verdiği rakamlar aşağıda verilmiştir:

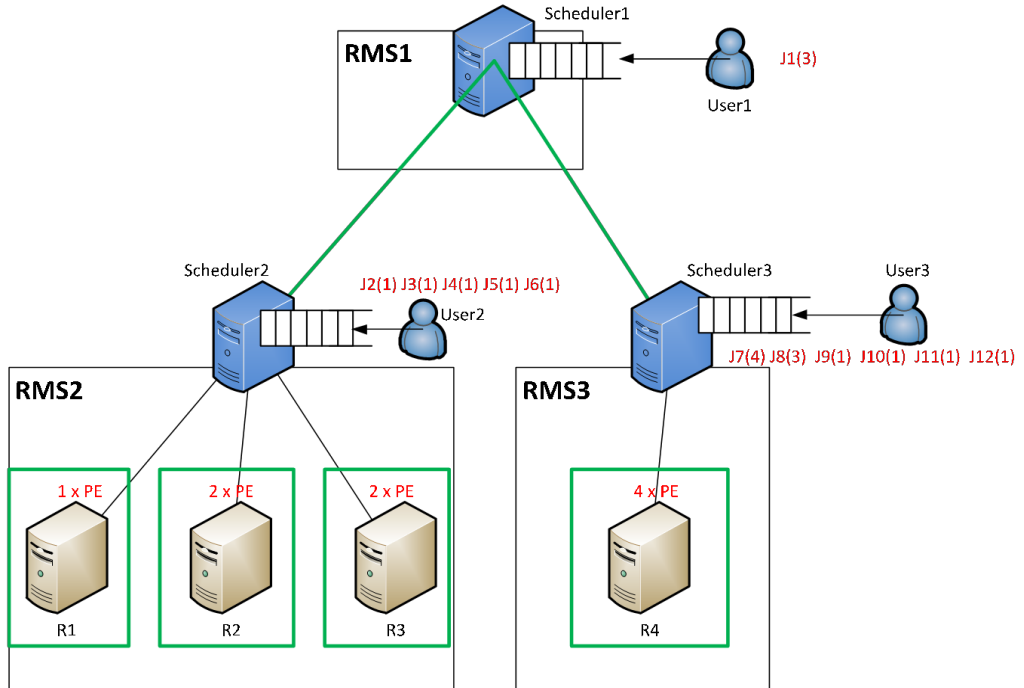
- AWT: 236.16666666666666
- MakeSpan: 301.0 (en uzun iş Job#09 için)
- Mean Job Length: 34.000.0 (408.000 /12 adet)



Şekil 5.29 : Deney2 sonucu iş-kaynak atama grafiği.

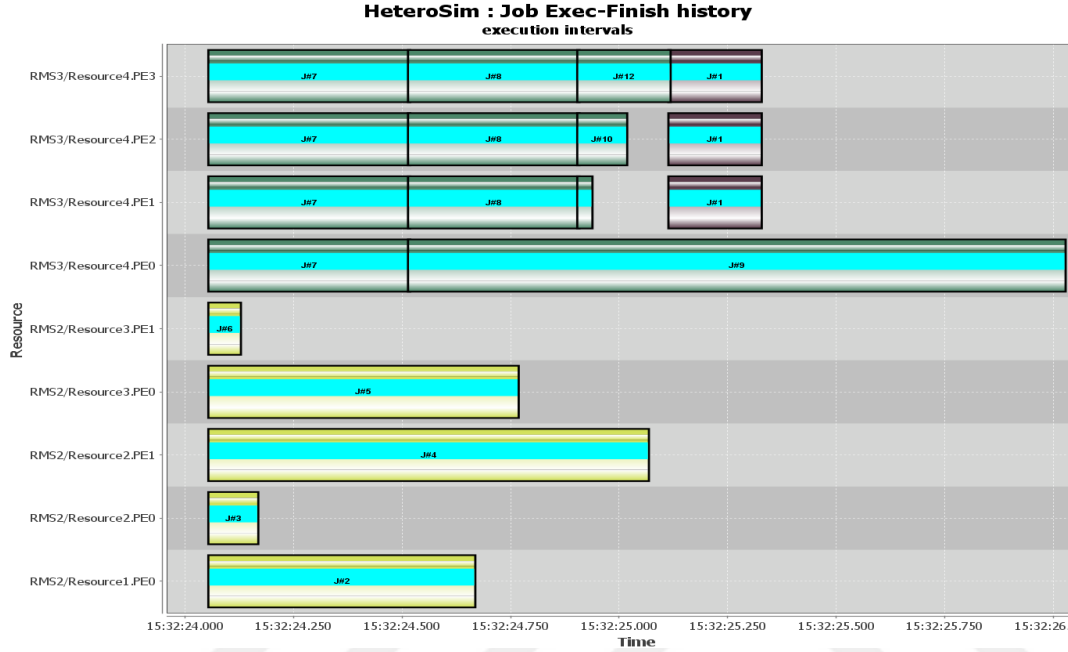
DENEY-3

Bu deneyde, R1, R2, R3 ve R4, dış RMS'lerin paylaşımına da açılmıştır. Önceki deneyden farkı, RMS2 ve RMS3 arasındaki peer arayüzünün kapatılmış olmasıdır. Bu yüzden, iki RMS birbirlerine iş gönderemeyecektir. Diğer bir fark, R1'in de dış paylaşımına açılmış olmasıdır (Şekil 5.30). Yine bütün kullanıcılar, testin başlaması ile birlikte, listelerindeki işleri aynı anda sisteme gönderirler.



Şekil 5.30 : Deney3 mimarisi.

Simülasyon süresince işlerin kaynaklarda çalışma zamanları Şekil 5.31’de gösterilmiştir. RMS3’ün kullanıcısı olan User3’e ait Job#9, Job#10, Job#11, Job#12, boş oldukları ve dış paylaşıma açık oldukları halde, RMS2’nin kaynaklarına (R1, R2, R3) atanmamışlardır. Çünkü RMS2 ile RMS3 arasındaki peer interface bu deneyde kapalıdır. Metrik grafikleri, önceki deneyden çok önemli fark göstermemiştir.



Şekil 5.31 : Deney3 sonucu iş-kaynak atama grafiği.

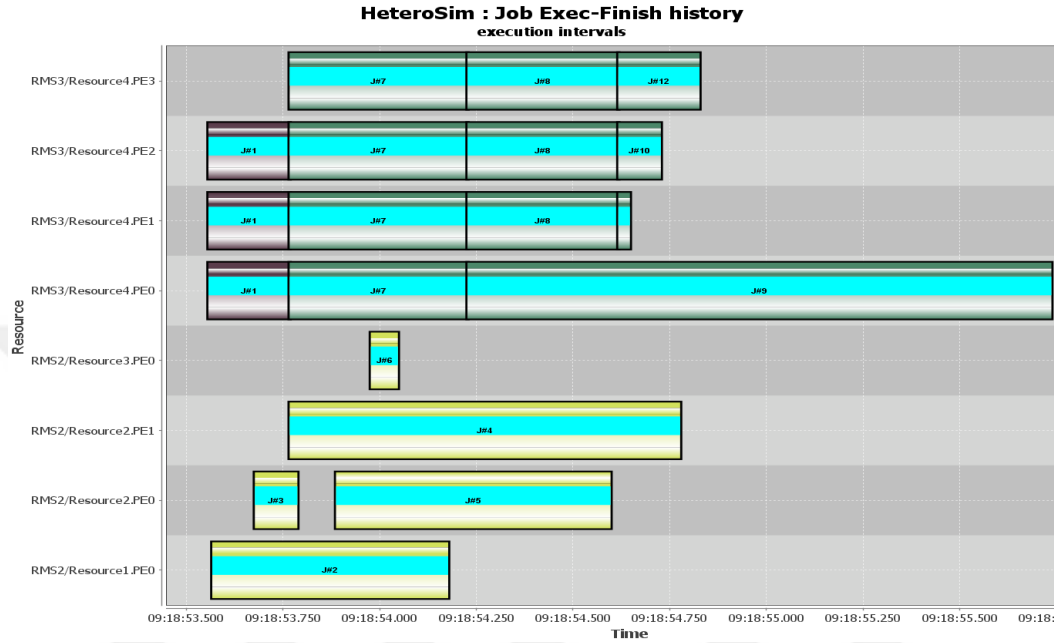
Simülasyon süresince işlerin kaynaklarda çalışma zamanları Şekil 5.31’de gösterilmiştir. RMS3’ün kullanıcısı olan User3’e ait Job#9, Job#10, Job#11, Job#12, boş oldukları ve dış paylaşıma açık oldukları halde, RMS2’nin kaynaklarına (R1, R2, R3) atanmamışlardır. Çünkü RMS2 ile RMS3 arasındaki peer interface bu deneyde kapalıdır. Metrik grafikleri, önceki deneyden çok önemli fark göstermemiştir. GridStatistics servisinin global metrikleri ise şu değeri almıştır:

- AWT: 215.16666666666666
- MakeSpan: 197 (en uzun path = Job#7 || Job#09 için)
- Mean Job Length: 34.000.0 (408.000 MI /12 adet)

Job#9 önceki deneylerden (R1, R2 ve R3’ün kullanılamaz hale gelmesi nedeniyle) daha geç başlamış ve bütün simülasyonun makeSpan değeri, Job#7’nin makeSpan değeri + Job#9 ‘un makeSpan değeri = 46 + 151 = 197 sn. olmuştur. Önceki deneylerden daha küçük sayı çıkmasının nedeni, Job#9’un bu kez iki kat hızlı olan R4 kaynağı üzerinde çalıştırılmış olmasıdır.

DENEY-4

YBH sistem mimarisi, bir önceki deneydeki ile aynıdır. Bu deneydeki tek fark, kullanıcıların işleri sisteme gönderme şeklidir. Kullanıcılar, simülasyon başlar başlamaz listelerindeki bütün işleri göndermek yerine, her işten önce, ortalaması $\lambda=10$ sn. olan *Poisson* dağılımına göre gecikmeler ekleyip göndermektedir.



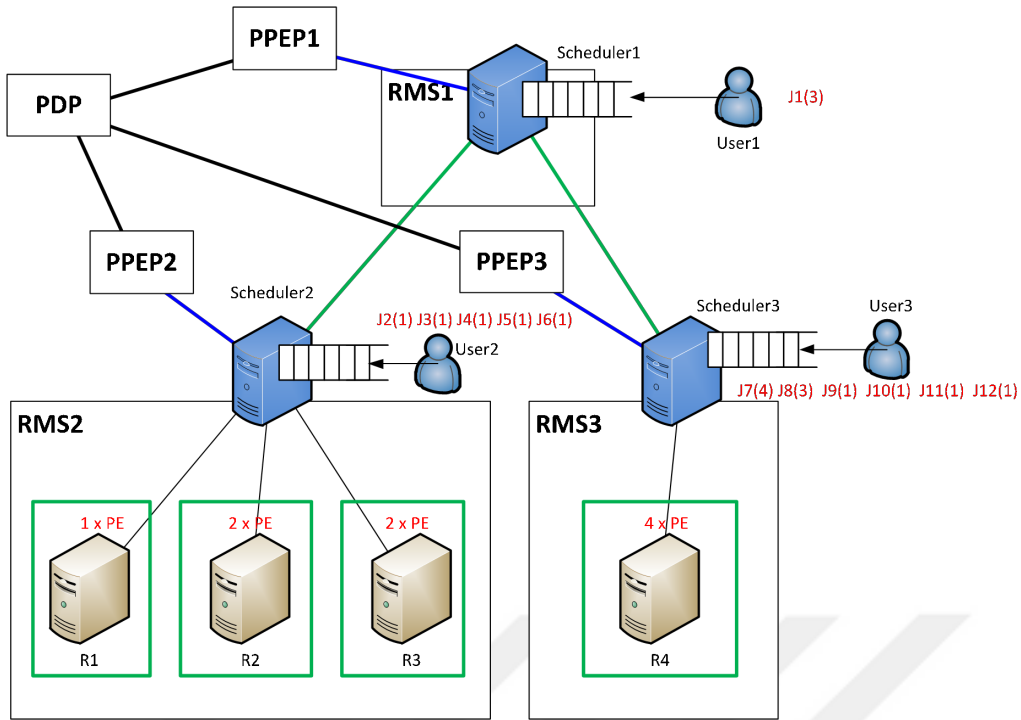
Şekil 5.32 : Deney4 sonucu iş-kaynak atama grafiği.

Deney4 Simülasyon süresince işlerin kaynaklarda çalışma zamanları Şekil 5.32’de gösterilmiştir. GridStatistics’in, deney sonundaki job süre bilgileri, global metrik değerleri aşağıda verilmiştir.

- AWT: 47.33333333333336 (Job7, Job8, Job9, Job10, Job11, Job12 bekleme süreleri toplamı /6 = (18+49+39+67+61+49)/6 = 283/6 = 47,3 sn.)
- MakeSpan: 227.0 (en uzun iş Job1 + Job7+ Job9 için = 9+21+46+151 = 227 saniye)
- Mean Job Length: 34.000.0 (408.000 /12 adet)

DENEY-5

Bu deneyin mimarisi, önceki deneyin mimarisine, Politika uygulama elemanlarının bağlanması ile genişletilmiştir. Her bir RMS’e bir Police PEP atanmıştır (Şekil 5.33). PDP’den tanımlanan erişim denetim politikalarının, kaynak atanmasında kullanılabildiğinin gösterilmesi amaçlanmıştır. Bütün kullanıcılar, testin başlaması ile birlikte, listelerindeki işleri, $\lambda=10$ saniye ortalaması olan poisson dağılımına uygun olacak şekilde sisteme gönderirler.



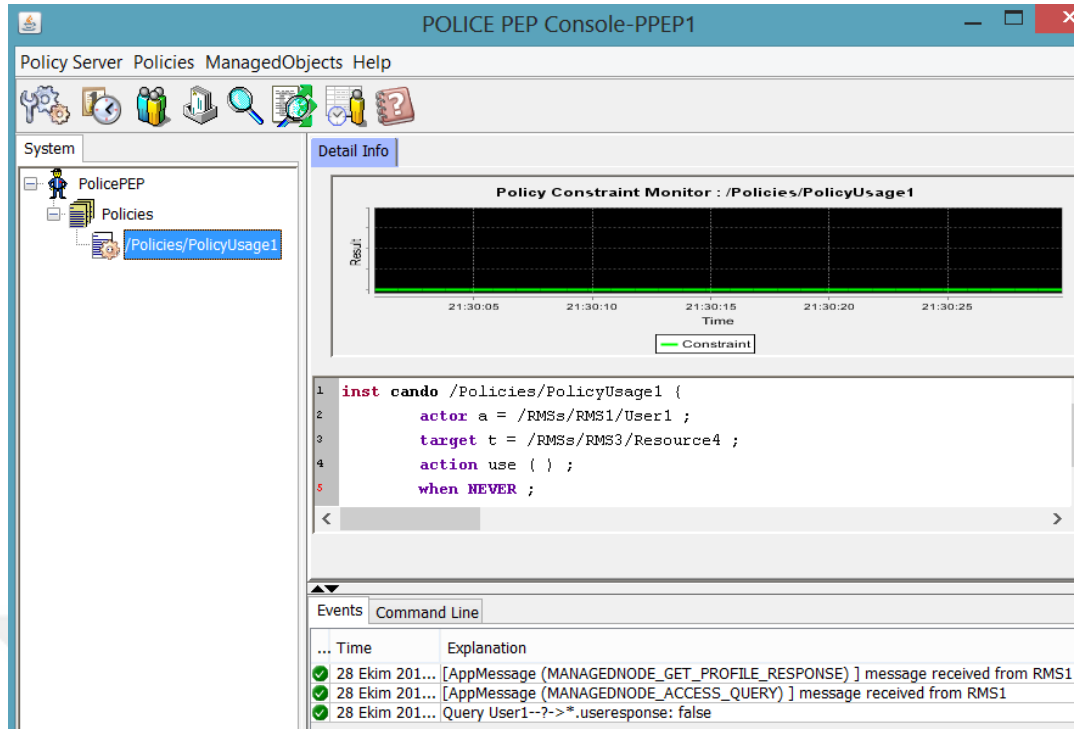
Şekil 5.33 : Deney5 mimarisi.

DENEY 5-1

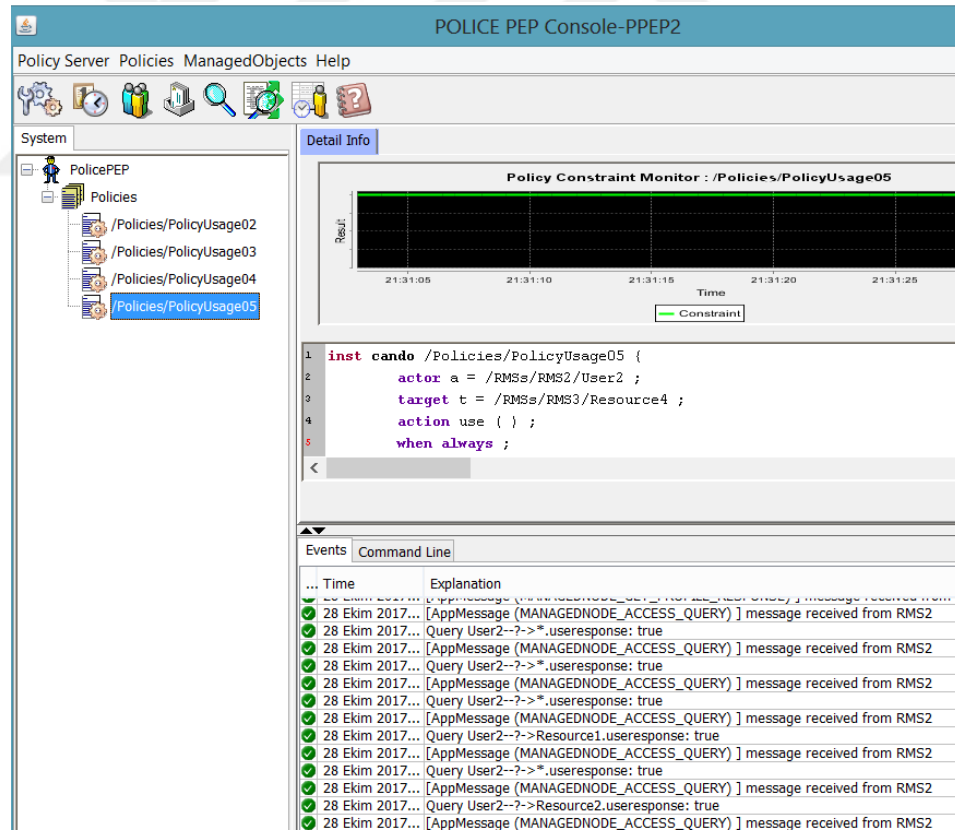
Bu alt deneyde Police yetki (*cando*) politikaları aşağıdaki gibi tanımlanmıştır.

- User1--- X ---> Resource4 (erişim izni yok)
- User2-----> Resource1, Resource2, Resource3, Resource4 (izin var)
- User3-----> Resource1, Resource2, Resource3, Resource4 (izin var)

Simülasyon sırasında, Threader ve Scheduler'ların PEP'lerden yeki sorguları, aşağıdaki gibi POLICE GUI'lerinden de izlenmiştir. PPEP1 ekranında, RMS1.Threader tarafından herhangi bir kaynağa izin var mı kontrolüne PPEP1'in yanıtı *false* olmuştur. Zira PolicyUsage1 politikası yasaklamıştır. Herhangi bir izin politikası yoktur. Dolayısıyla User1'in Job#1'i simülasyonda kendisine kaynak bulamayıp çalışma fırsatı bulamamıştır. Politikayı tanımlarken *actor = User1*, target olarak ise *Resource4.use()* metodu kullanılmıştır. PPEP2 ekranında, RMS2.Threader ve Scheduler tarafından yürütülen kontrollere, PPEP2'in yanıtı *true* olmuştur. Zira PolicyUsage02, PolicyUsage03, PolicyUsage04, PolicyUsage05 politikaları kaynaklara erişim izni vermiştir. Dolayısıyla User2'nin bütün iş yükleri simülasyonda kendisine kaynak bulup çalışmıştır. User1 ve User2'nin Resource4'e erişiminin, tanımlanan politikalarla uyum içinde gerçekleştiği, Şekil 5.34 ve Şekil 5.35'de verilen Police Kullanıcı arayüzlerinden de izlenmiştir.



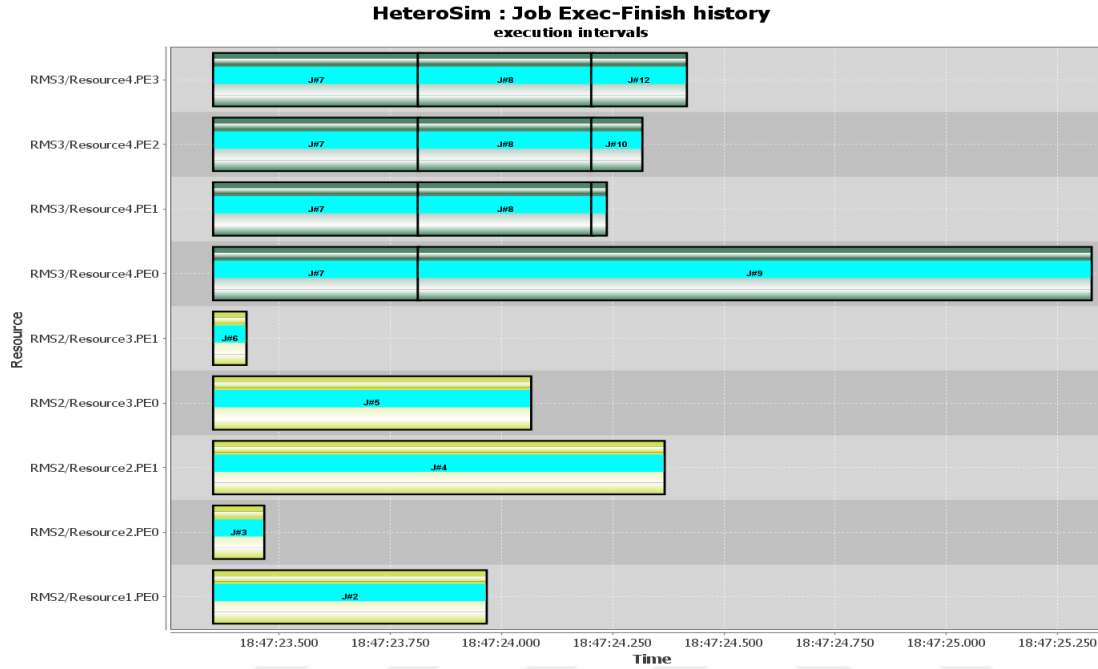
Şekil 5.34 : Deney5 Resource4-User1 erişim politikası.



Şekil 5.35 : Deney5 Resource4-User2 erişim politikası.

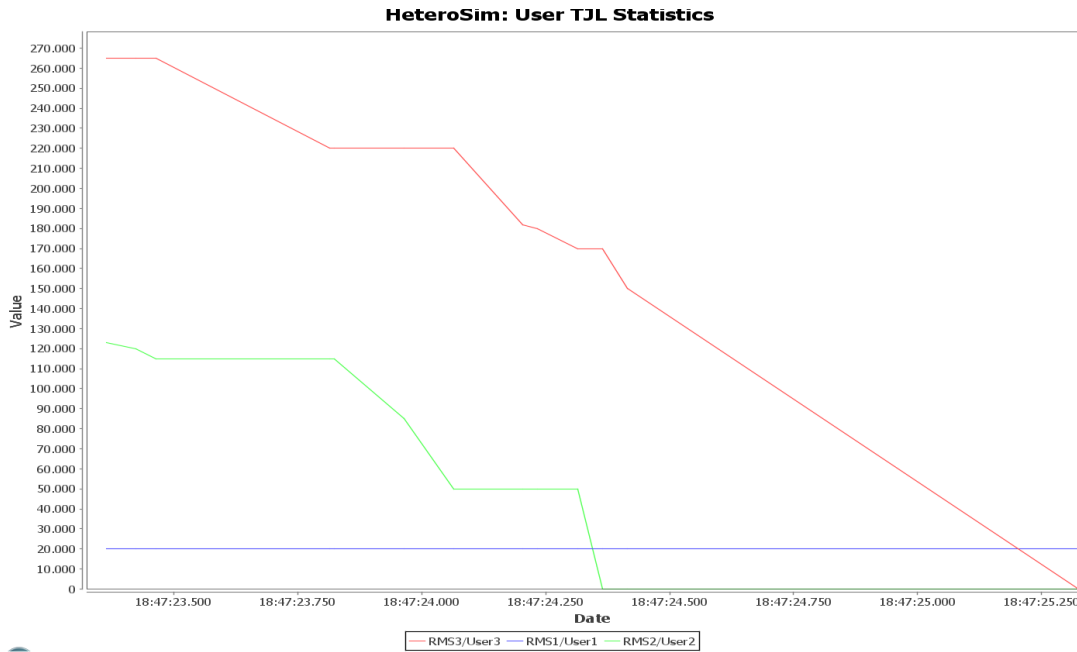
PPEP3 ekranında, RMS3.Threader ve Scheduler tarafından herhangi bir kaynağa izin var mı kontrolüne PPEP2'in yanıtı *true* olmuştur. PolicyUsage02, PolicyUsage03,

PolicyUsage04, PolicyUsage05 politikaları kaynaklara erişim izni verdiği için, User2'nin bütün iş yükleri simülasyonda kaynak bulup çalışmıştır. RMS2 ile RMS3 arasında peer arayüzü kapalı olduğu için, beklendiği gibi birbirlerinde iş çalıştırmamışlardır (Şekil 5.36).



Şekil 5.36 : Deney5 sonucu iş-kaynak atama grafiği.

Toplam İş Boyu metriği (TJL), User1 iş yükü boyu simülasyon sonuna kadar tüketilmediği için hep sabit (20.000 MI olarak) kalmıştır. Bu durum Şekil 5.37'de verilen metrik grafiğinden de izlenebilmiştir.



Şekil 5.37 : Deney5 sırasında Toplam İş Boyu (TJL) metriği grafiği.

GridStatistics servisinin, kořturma sonunda verdiđi job süre bilgileri, global metrik deđerleri ve bu deđerlerin teorik olarak sađlamaları ařađıda verilmiřtir:

- AWT: 231.0833 (Job1 + Job7.wait + Job8.wait + Job9.wait + Job10.wait + Job11.wait + Job12.wait /7)
- MakeSpan: 192.0 (en uzun iř Job7+Job9 iin = 192 saniye)
- Mean Job Length: 34.000.0 (408.000 /12 adet)

DENEY 5-2

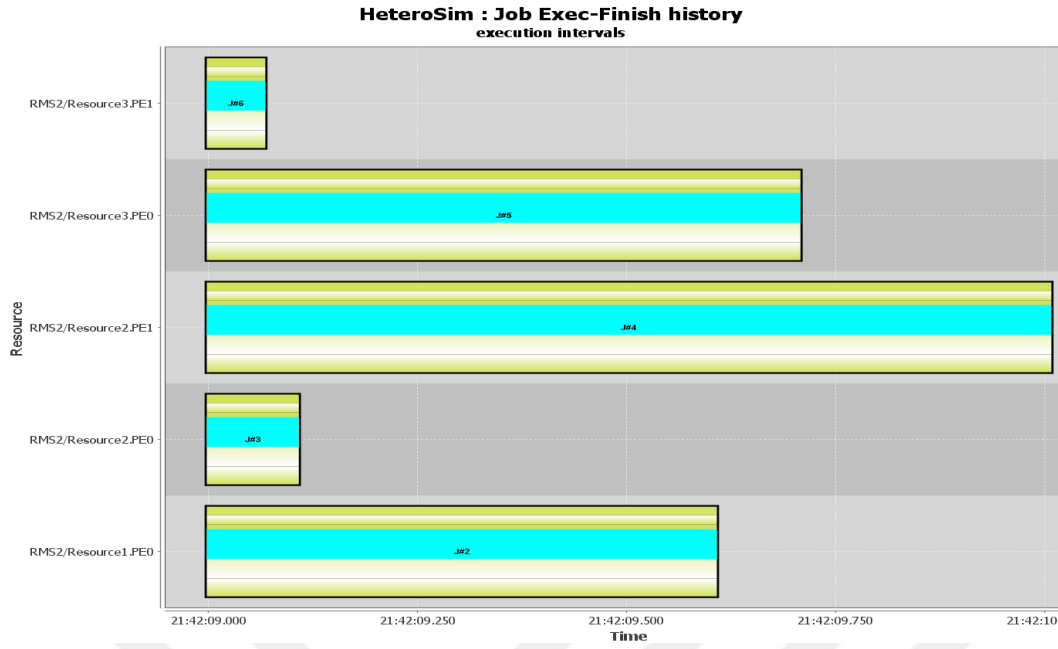
Bu alt deneyde Police yetki (cando) politikaları ařađıdaki gibi tanımlanmıřtır.

- User1--- X ---> Resource4 (eriřim izni yok)
- User2-----> Resource1, Resource2, Resource3, Resource4 (izin var)

User3 iin herhangi bir politika tanımlanmamıřtır. Police aracının, “*aksi belirtilmedike her řey yasaklıdır*”, tasarım prensibi geređi, User3’ün kaynak eriřim istekleri, bir izin politikası bulunmaması nedeniyle reddedilmiřtir. Simlasyon sırasında, Threader ve Scheduler’ların PEP’lerden yeki sorguları, ařađıdaki gibi POLICE GUI’lerinden de izlenmiřtir. PPEP1 ekranında, RMS1.Threader tarafından herhangi bir kaynađa izin var mı kontrolne PPEP1’in yanıtı false olmuřtur. Zira PolicyUsage1 politikası yasaklamıřtır. Dolayısıyla User1’in Job#1’i simlasyonda kendisine kaynak bulamayıp alıřma fırsatı bulamamıřtır.

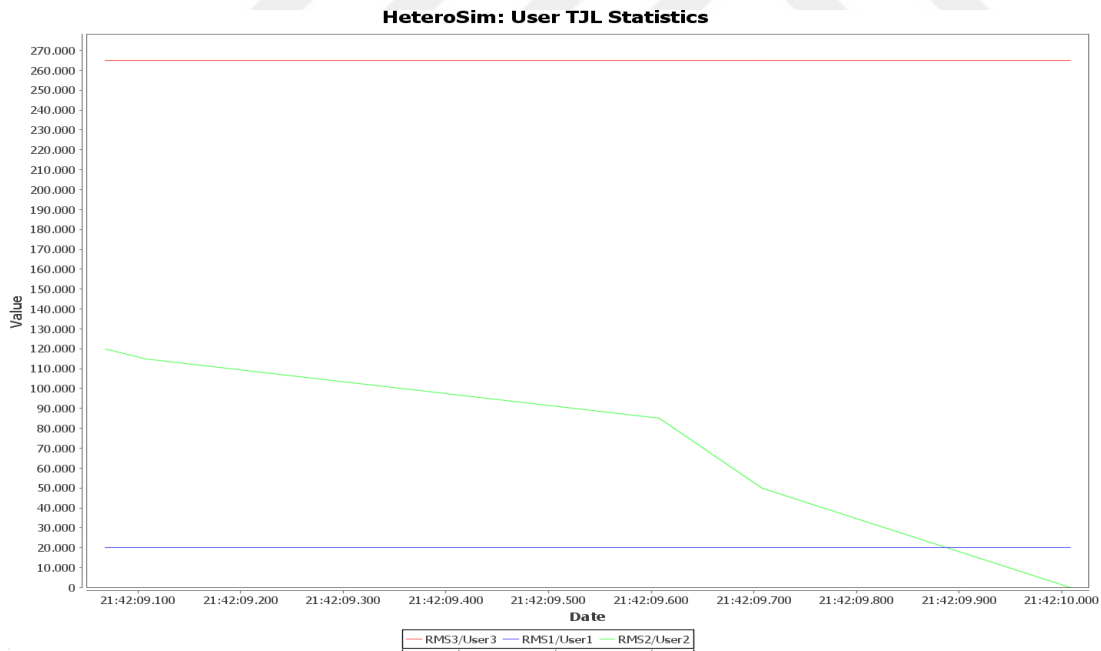
PPEP2 ekranında, RMS2.Threader ve Scheduler tarafından herhangi bir kaynađa izin var mı kontrollerine PPEP2’in yanıtları, ařađıdaki GUI’den de grldđ zere, **true** olmuřtur. Zira PolicyUsage02, PolicyUsage03, PolicyUsage04, PolicyUsage05 politikaları kaynaklara eriřim izni vermiřtir. Dolayısıyla User2’nin btn iř ykleri simlasyonda kendisine kaynak bulup alıřmıřtır.

PPEP3 ekranında, RMS3.Threader tarafından herhangi bir kaynađa izin var mı kontrolne PPEP3’in yanıtı false olmuřtur. Zira herhangi bir politika tanımlı deđildir. Dolayısıyla User3’n iř ykleri kaynak bulamayıp alıřma fırsatı bulamamıřtır. řekil 5.38’deki atama grafiđinde, sadece User2 iř yklerinin alıřtırıldıđı grlmřtir.



Şekil 5.38 : Deney5 sonucu kısıtlayıcı politikalar ile iş-kaynak atama grafiği.

Toplam Job Length metriği (TJL), User1 ve User3'ün iş yükü boyları simülasyon sonuna kadar tüketilmediği için hep sabit (sırasıyla 20.000 MI ve 265.000 MI olarak) kalmıştır. Bu durum aşağıdaki metrik grafiğinden de izlenebilmiştir.



Şekil 5.39 : Deney5 sırasında kısıtlayıcı politikalar ile TJL metriği grafiği.

GridStatistics servisinin, koşturma sonunda verdiği job süre bilgileri, global metrik değerleri ve bu değerlerin teorik olarak sağlamaları aşağıda verilmiştir:

- AWT: $201.0 ((\text{Job1} + \text{Job7.wait} + \text{Job8.wait} + \text{Job9.wait} + \text{Job10.wait} + \text{Job11.wait} + \text{Job12.wait})/7)$

- MakeSpan: 101.0 (en uzun iş Job4 için = 101 saniye)
- Mean Job Length: 34.000.0 (408.000 /12 adet)

5.3.4.2 HeteroSim ölçeklenebilirlik ve performans analizi

HeteroSim performans analizi, bu bölümde açıklanmış ve [11] makalesinde yayımlanmıştır. Ölçeklenebilirlik en önemli tasarım kriterleri olarak ön plana çıkmıştır. Simülasyonlar, milyonlarca prosesin bir makinada çalıştırılması için mekanizmalar sağlamadıkça ölçeklenebilir sayılamazlar (Dağıtık simülasyonla prosesler birden çok makineye dağıtılamıyor varsayımıyla). Zira standard VM teknolojileri, bir makinada ancak onlarca VM etkin olarak çalıştırılabildiği için uygun düşmez diye bilinir.

Thread tabanlı olarak simülasyon, varlıklarının eş zamanlı davranışlarının modellenmesi amacıyla çoğu araç tarafından (özellikle DES tabanlı olanlar) kullanılsa da işletim sistemlerinde thread sayısı sınırlamaları, thread bağlam anahtarlamaları nedeniyle oluşan yükler, İş Sıralayıcının deterministik olmayan davranışları, threadlerin yığın bellekleri ihtiyacı nedeniyle dikkatli kullanılması gereken bir yöntemdir. Çoğu Java tabanlı gerçekleştirme gibi, HeteroSim’de de simüle edilen proseslerin herbiri bir Java Thread olarak çalışır. Standart thread teknolojisi ile ancak onbinlerce süreç simüle edilebilir. Bu sonuca ulaşan bir Grid simülatörleri analizi, [196] makalesinde verilmiştir. Makalede yazarlar, kendi geliştirdikleri tek lifli GES aracı ile çoklu-lifli SimGrid ve GridSim simülatörlerini karşılaştırmışlar. GridSim, 10.000 *User* içeren simülasyon için en az 32.000 adet thread oluşturmak zorundadır. HeteroSim, aynı thread sayısına yaklaşık 20.000 kullanıcıda, SimGrid ise 12.000 kullanıcı sayısında ulaşır. SimGrid, standart işletim sistemi threadleri yanında Linux kullanıcı uzayında çalışan (*ucontext*) threadleri ve *continuation denilen* (cooperative, light-weight, non-preemptive threads) seçeneği sunduğu için bu limiti aşabilmektedir.

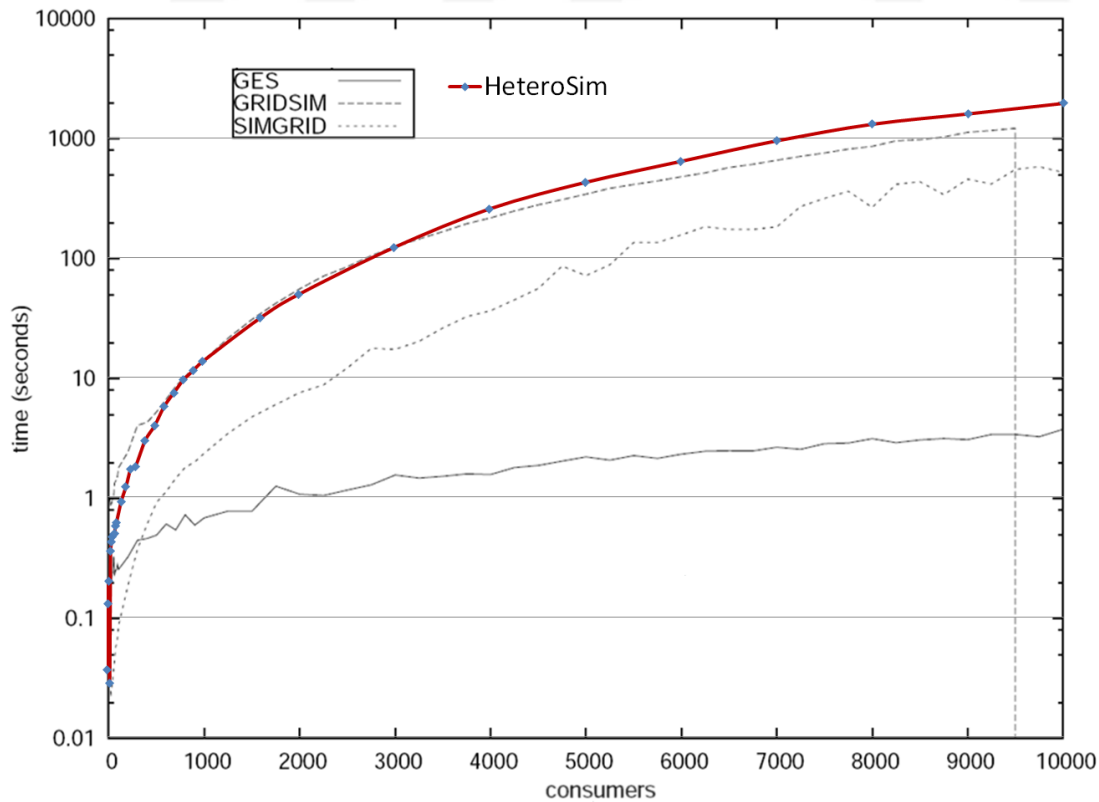
HeteroSim’in performansını incelemek için, [196] makalesinde tasarlanmış olan ölçeklenebilirlik ve performans testleri tekrarlanmıştır ve sonuçları paylaşılmıştır.

Makalede yayınlanan bütün testler, Antwerp Üniversitesi’ndeki CalcUA kümesinde çalıştırılmıştır. Bu kümede, 256 adet, 64-bit Linux işletim sistemli, Opteron 250 işlemcili bilgisayar bulunur. Kullanılan bilgisayarlar, 8GB RAM’e sahiptir. Java olarak ise Sun JVM 1.6.0 kullanılmıştır. Makalede, testlerde çalışan Java sürecinin parametrelerini 1 saniye periyotlarla örnekleyen grafikleri paylaşmışlardır.

Makaledeki simülasyonlarda, her bir kaynağa (Resource/Provider) en fazla 10 tane kaynak kullanıcısı (User/Consumer), düşmüştür. Böylece, kuyruklardaki yığılmalar, karmaşık topolojiler ve simülasyon varlıklarının etkileşiminden dolayı oluşabilecek yan etkiler minimize edilerek simülasyonun motorunun çalışmasına odaklanılmıştır. Testler, HeteroSim ile de gerçekleştirilip sonuçları, makalede yayınlanmış olan grafiklerin üzerine eklenerek bu bölümde verilmiştir. Bizim testlerimiz, 64-bit Windows 7 işletim sistemli, 8 GB RAM'e ve Intel i-7 4500U 1.8 GHz işlemciye sahip bilgisayarda, 1.7.0_45 sürümlü Oracle Java ortamında çalıştırılmıştır. Thread'ler için yığın boyu 128 KB olarak ayarlanmıştır.

Test-I. Genel Ölçeklenebilirlik

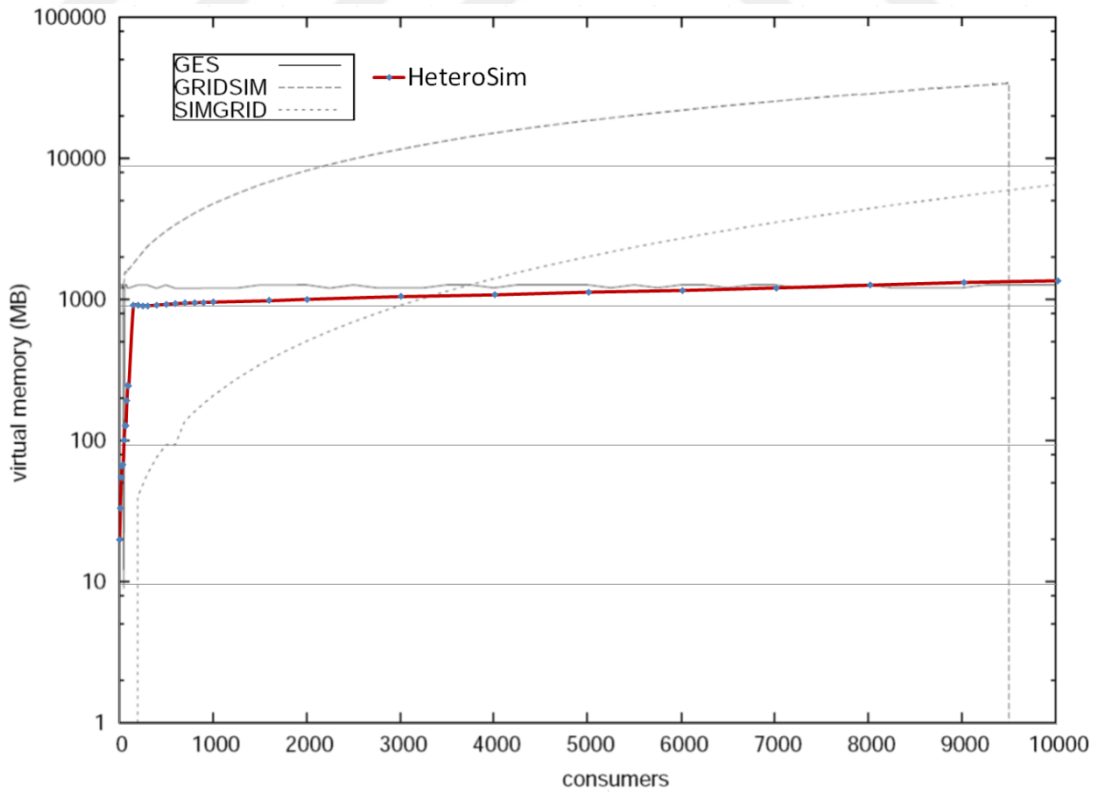
Bu testte, kullanıcı sayısı 1'den 10.000'e kadar artırılarak simülasyon tekrar çalıştırılmıştır. Her bir kaynağa iş gönderen kullanıcı sayısı 10 adeti geçmeyecek şekilde kaynak sayısı da artırılmıştır. Logaritmik ölçekte çizilen sonuç, Şekil 5.40'dadır. Grafikte de görüldüğü üzere, GES en iyi ölçeklenebilen araçtır. Simülasyon süresindeki farklar oldukça fazladır. HeteroSim, GridSim altyapısı üzerine inşaa edildiği için süreler yakındır. GridSim, 10.000 kullanıcı testi yürütemez. Zira, 32.000 adet natif thread'den fazlası gerekmektedir ama Linux çekirdeğinde bu eşiğin özel



Şekil 5.40 : Simülasyon süresinin kullanıcı sayısına göre değişimi.

olarak değiştirilmesi gerekir. Windows7 işletim sisteminde thread sayısı ancak kullanılabilir bellek alanıyla sınırlıdır. Bu yüzden bizim gerçekleştirdiğimiz testlerde Thread sayısında bu eşiğe takılma olmamıştır. GES, simülasyon varlıklarını thread olarak modellemediği için thread sayısına duyarlılığı yoktur. SimGrid ise 12.000 kullanıcı sayısında 32.000 thread sayılarına ulaşır.

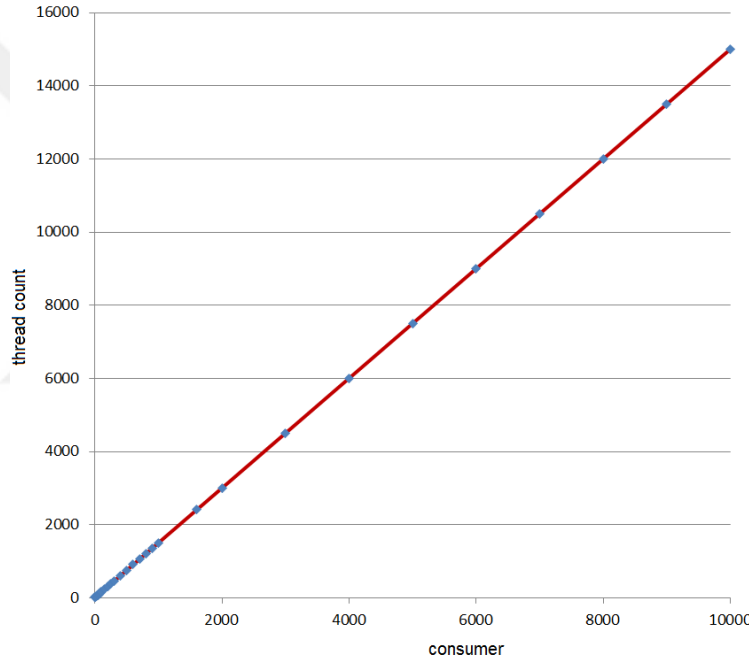
Ölçeklenebilirliğin önündeki sınırlamalar (thread sayısı ve sanal bellek) makalede incelenmiştir. Şekil 5.41’de araçların sanal bellek kullanım performansları gösterilmiştir. Aynı makalede sanal ve fiziksel bellek kullanım rejimleri, 32-bit makinadaki Java sınırlamaları üzerinden incelenmiştir. Proseslerin 32-bit işletim sistemlerinde 4GB sanal bellek sınırlaması vardır. 64-bit Linux bilgisayararda ise bu sınır, fiziksel bellek boyu + swap boyutu ile belirlenir. Normal Java Thread yığınının boyu 320KB (32-bit makina), 1024KB (64-bit makine)’dir. Bu değer JVM parametresi ile minimum 64 KB’a kadar düşürülebilir. Linux *pthread*’lerde 56KB’e kadar, *ucontext* tipli threadlerde ise daha da küçük olabilir. Makalede elde edilen sonuçlarda da gösterildiği gibi, araçlar (özellikle 1 MB’lık ön tanımlı yığın boyu nedeniyle) rezerve ettikleri sanal belleklerin çok azını kullanırlar. Ayrıca yığın, öncelikle sanal bellekte oluşturulduğu için thread kullanımının fiziksel belleğe etkisi doğrudan



Şekil 5.41 : Sanal bellek kullanımının kullanıcı sayısına göre değişimi.

olmayabilir. Bu sebeple testlerimizde java yığın boyu küçük tutulmuştur. Makalede 64 KB olan yığın boyu, testlerimizde 128 KB olarak kullanılmıştır. Sanal bellek kullanımımız, SimGrid ve GridSim'den daha yavaş eğitimle artmıştır.

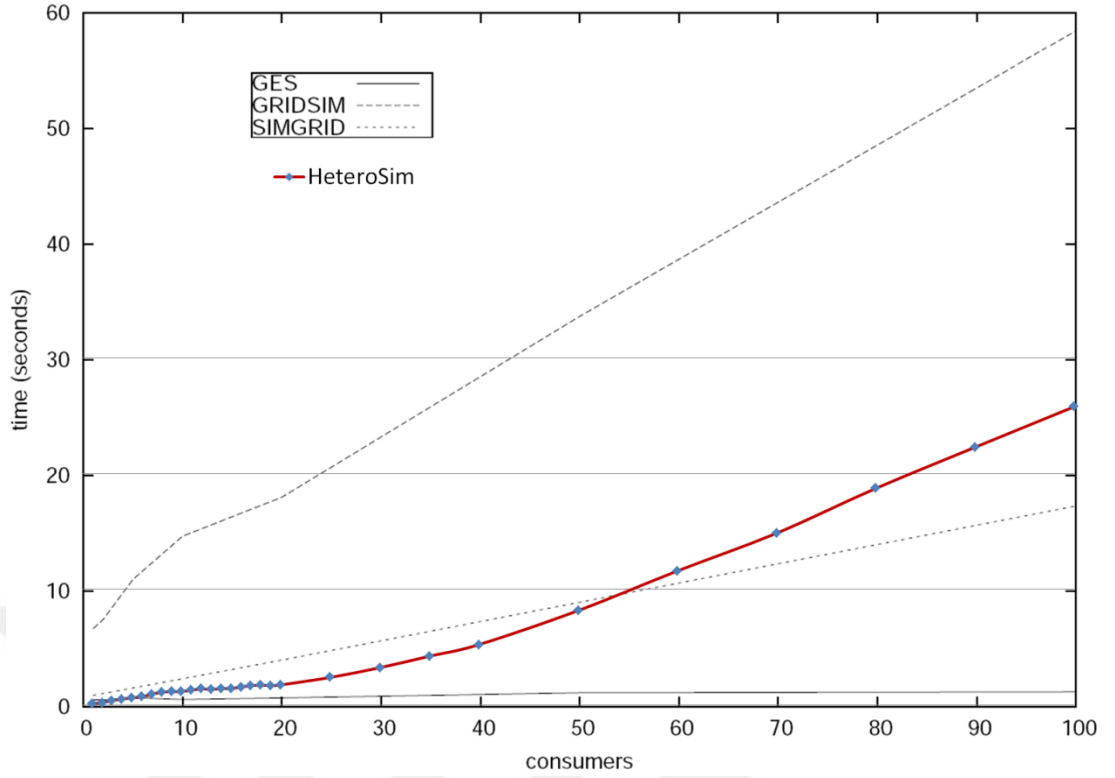
HeteroSim'in Test-I konfigürasyonundaki simülasyonlarda, thread sayısının kullanıcı sayısına göre lineer ilişkisinin formülü ise yaklaşık olarak $\text{Thread\_sayısı} = 1,5 \times \text{Kullanıcı\_sayısı}$ olarak ifade edilebilir olduğu görülmüştür. Zaten, test yürütülürken, 10.000 user, 1.000 Kaynak yanında, her kaynak için 1 Scheduler, 1 Treader, 1 Dispatcher, 1 LIS ve simülasyon geneli servisleri için 1 GIS, 1 Shutter olarak düşünülürse yaklaşık 15.000 Thread oluşur (Şekil 5.42).



Şekil 5.42 : Maksimum Thread sayısının kullanıcı sayısına göre değişimi.

Test-II. İşyükü Ölçeklenebilirliği

Bu test ile iş yükü sayısının simülasyon süresine etkisi incelenmiştir. Testte, her kullanıcı, 100 adet iş yükü gönderir. Kullanıcı sayısı 1 ile 100 arasında artırılarak test tekrarlanır. Yine önceki testteki gibi her bir kaynağa en fazla 10 kullanıcının iş göndermesi sağlanacak şekilde kaynak sayısı da artırılır. Kendi sonuçlarımızın da eklendiği makaledeki grafik Şekil 5.43'de gösterilmiştir.



Şekil 5.43 : Simülasyon süresinin kullanıcı sayısına göre değişimi.

Makalede ifade edildiği üzere, GES'in iş sayısına dayanıklılığı en yüksek çıkmıştır. DES yöntemi kullanan araçlarda ise simülasyon süresi iş yükü sayısı ile lineer olarak ölçeklenebilir. Ayrık zaman için ise tersine, simülasyon süresi iş yükünün boyu ile orantılı artar. Bu iki modelden hangisinin daha uygun olduğu, modellenen zaman çözünürlüğü hedefine göre belirlenir. Protokollerin analizi için yükek çözünürlük gerekirken iş yürütmenin modellenmesi için gerekmez.

DES, genelde çok-lifli gerçeklemeyi gerektirir (Çünkü bağımsız çalışan prosesler arasında dolaşan olayların mantıksal modellenmesine dayanır). Ama bunun mecburi olmadığı [161] ve [200] gibi çalışmalarda gösterilmiştir. Tek lifli olarak da DES mümkündür ve daha iyi performans gösterebilir. Bu testte, HeteroSim, GridSim'den daha iyi performans göstermiştir. 50 kullanıcı sayısına kadar SimGrid'den de iyi olmasına karşın, artan kullanıcı sayısı ve iş yükü ile birlikte özellikle İstatistiksel hesaplamaların artan yükü nedeniyle SimGrid'in gerisinde kalmıştır.

Geliştirilen HeteroSim simülasyon ortamımız da benzer olarak Java thread kullanım limitleri ile sınırlıdır. Daha az sayıda iş içeren simülasyon oturumlarında sorunsuz çalışsa da birkaç yüzbin adet işi içeren simülasyonlarda yüksek bilgisayar kaynağı ihtiyacı oluşmakta ve simülasyon yavaşlamaktadır. Bu ise aracımızın ölçeklenebilirliği

konusunda iyileştirmeye açık noktalar olduğunu gösterir. Bunların yanında, çalışma izlerini canlı olarak bellekte biriktiren istatistik servisi de çok büyük simülasyon oturumlarında yavaşlamaya sebep olabilmektedir.

Gerçek iş yükü verileri ve metrikler

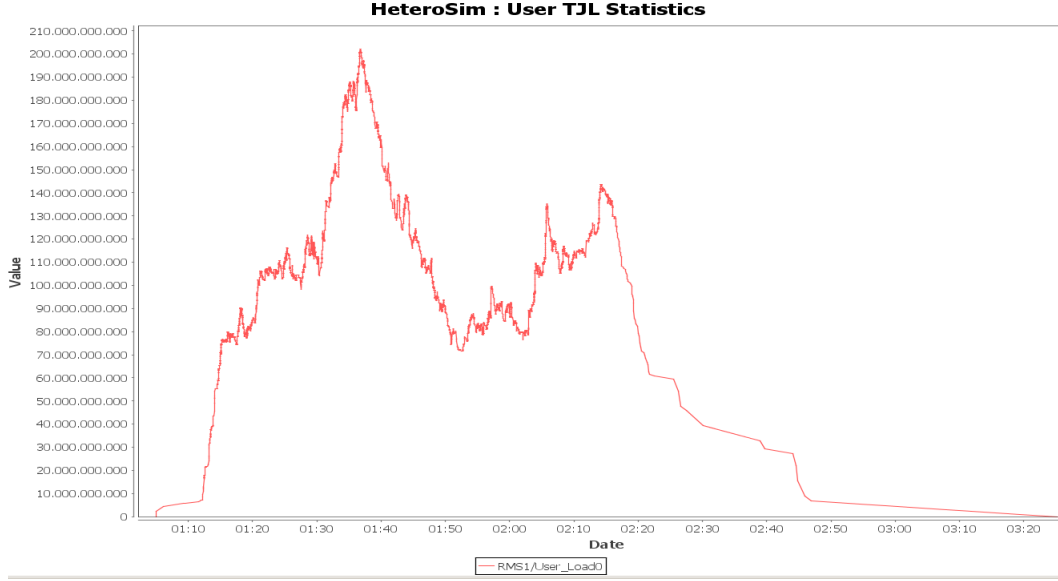
HeteroSim'in büyük adette iş yükü davranışını incelemek üzere, gerçek sistemlere ait çalışma kayıtları, HeteroSim'e de gönderilerek deneyler yapılmıştır. Simülasyonların sonuçlarının karşılaştırılabilir olması için aynı yükün farklı deneylerde kullanılması gerekir. Literatürdeki çalışmalar, bazı hesaplama merkezlerinden (NERSC- at Lawrence Berkeley National Laboratory, Lawrence Livermore National Laboratory, San Diego Supercomputer Center vb.) aldıkları gerçek iş yükü verilerini kullanırlar. İş yükü izi (traces) bilgisi alınabilecek halka açık merkezlerden birisi (<http://www.cs.huji.ac.il/labs/Parallel/workload/>) arşividir. Bu veriler, birkaç aylık periyotta, birkaç bin iş yüküne ait tüm bilgileri içerirler. 430 düğümlü IBM RS6000/SP paralel bilgisayarı olan *Cornell Theory Center*'ın iş yükü izleri halka açıktır.

Bununla birlikte, gerçek iş yüklerini kullanmada bazı zorluklar vardır. Alınan bu büyük veriyi kendi sistemimizin yapısına ve anlayabileceği formata çevirmek gerekir. Gerçek loglardan sentetik iş yükleri türetilir. Böylece iş yüklerinin sisteme gelişleri arasındaki süre ve büyüklükleri, çeşitli olasılık dağılımlarına göre tekrar düzenlenip simülasyonlar çalıştırılabilir. Örnek çalıştırmalar bu bölüm sonunda paylaşılmıştır.

Bu verileri evrensel YBH modelimizin simülasyonlarında kullanıp bazı metriklerle sistem performansı ve kullanıcı memnuniyetinin ölçülmesi hedeflenmiştir.

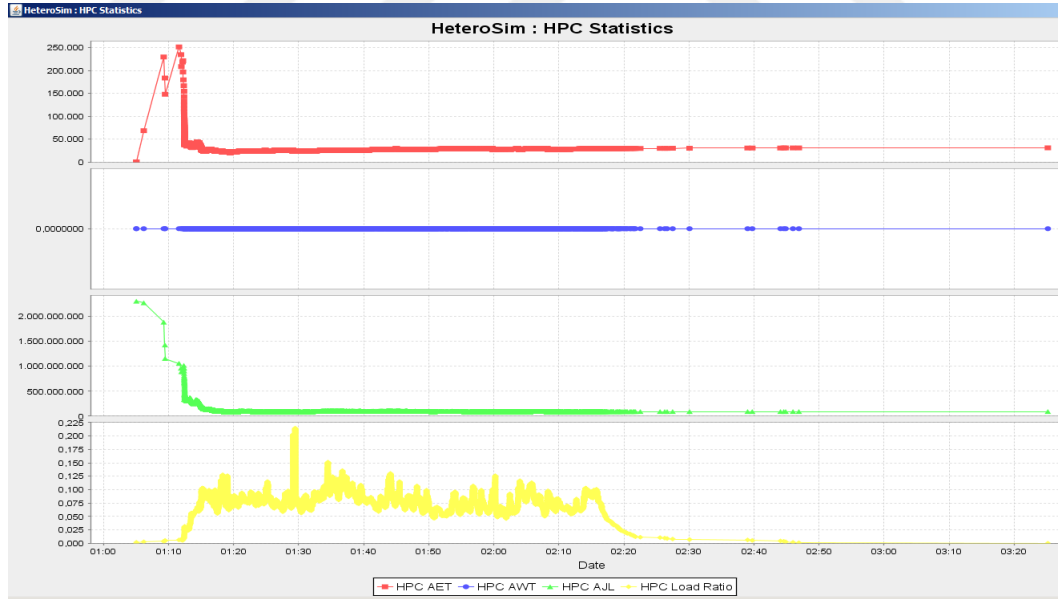
GridSim'in dosyadan gerçek sistemde yürütülmüş işlerin çalışma izlerini yükleme modülü, *WorkLoadTrace*, HeteroSim aracına entegre edilmiştir. Dosyadan okuduğu kaynak tanımlarından simülasyona kaynak modelini ekleyen modül de HeteroSim'e entegre edilmiştir. Kendisine verilen *WLTrace* dosyasını sisteme gönderecek ve paralel olarak işlerin sonuçlarını alabilecek *UserEntity* kodu yazılmıştır.

SDSC Blue-Horizon isimli süper bilgisayarda 2006 yılında çalışmış iş izi HeteroSim'de çalıştırılarak elde edilen metrikler aşağıdaki grafiklerde verilmiştir. Yaklaşık 96.000 adet job, 144 düğümlü makine içeren simülasyonda (her düğüm 8 adet 375 MHz CPU ile SMP olan ve 4 GB düğüm belleğini paylaşan) yürütülmüş, sonuçları ise Şekil 5.44, Şekil 5.45 ve Şekil 5.46'da gösterilmiştir. Simülasyonlar, 64-bit Intel i7 işlemcili, 4 GB belleği olan Windows makinada yürütülmüştür.

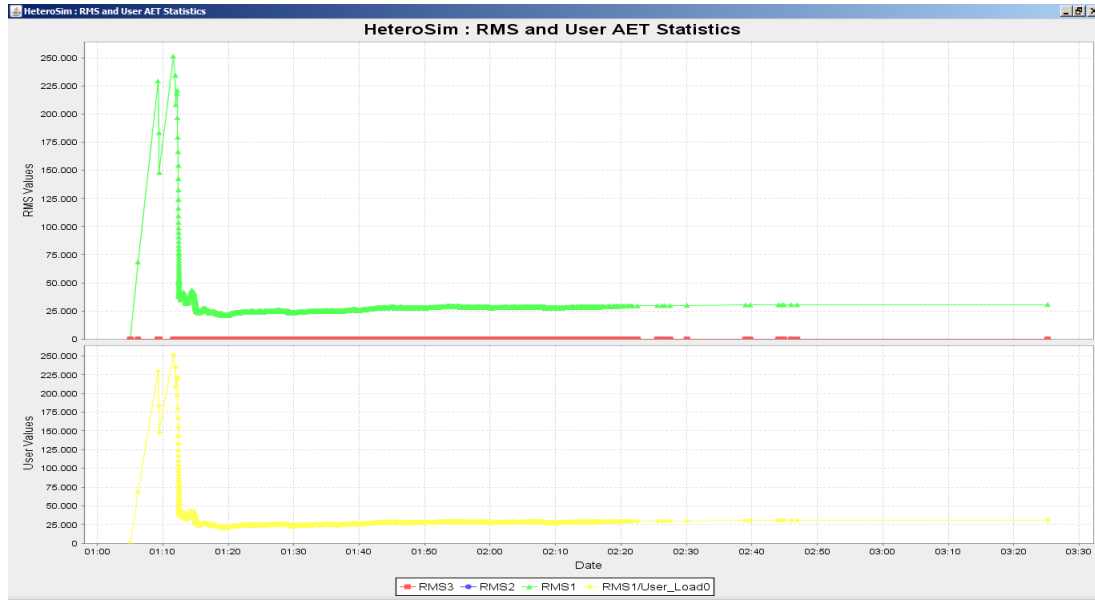


Şekil 5.44 : SDS Blue Gene iş izi, HeteroSim yürütmesi (TJL metriği).

Bu çıktıların incelemesinden, simülasyonun başlatılmasından, görevlerin etkin olarak yürütülmesinin başlamasına kadar, işlerin sisteme ilk gönderilmesine ait Java Thread'lerinin yoğun CPU kullanımı nedeniyle oluşan atalet görülebilir. Bu durum öncelik mekanizmaları ile düzeltilebilir.

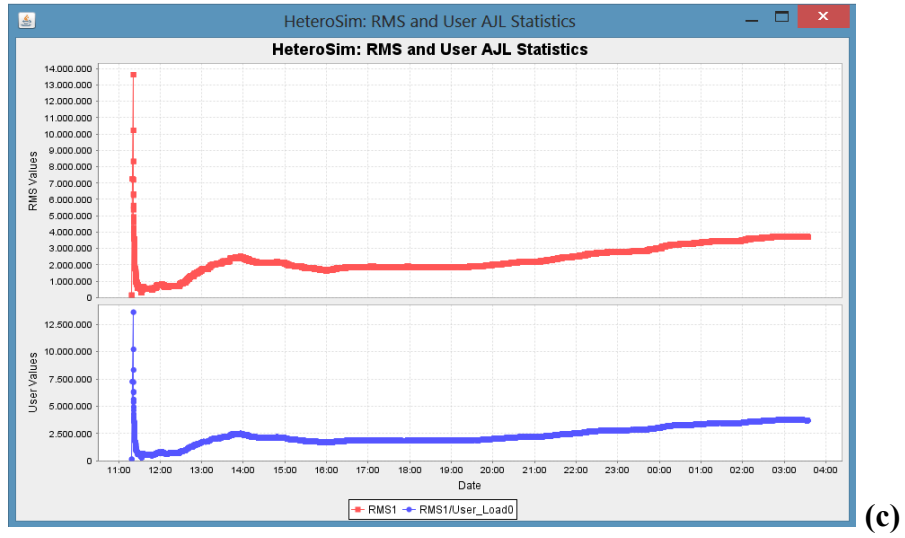
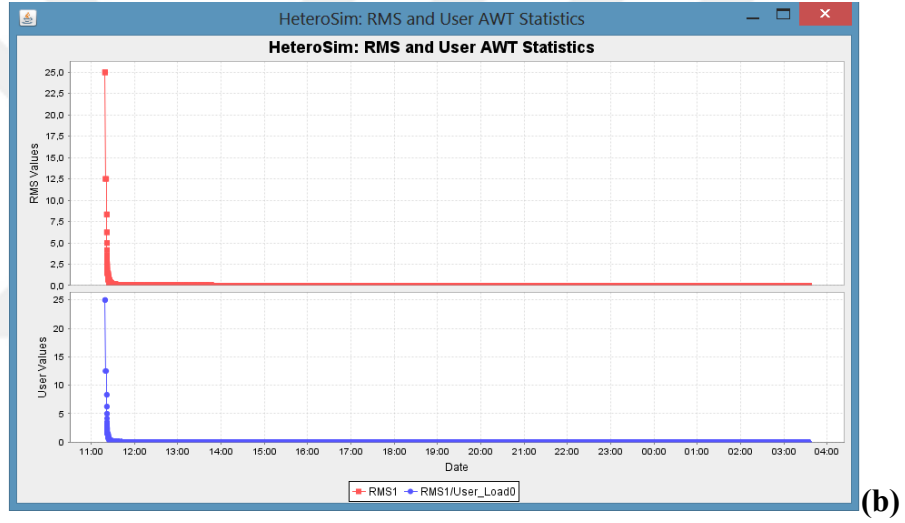
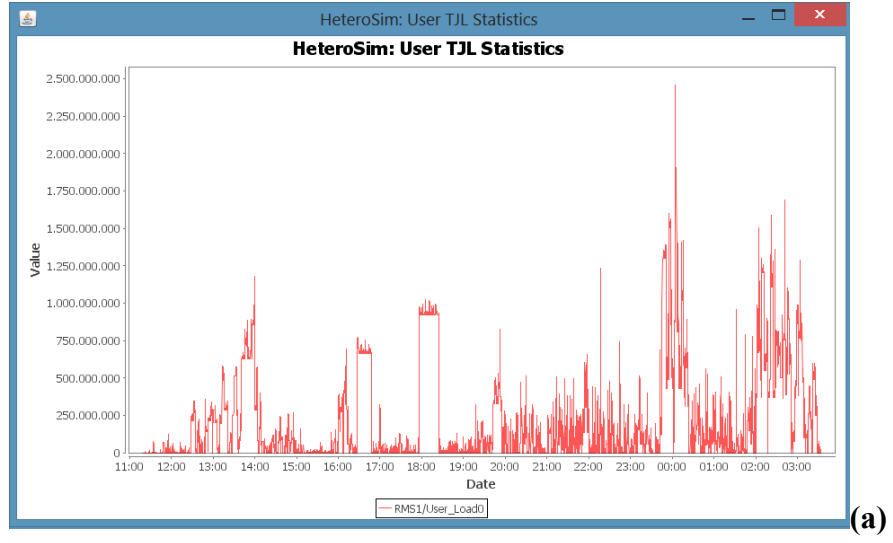


Şekil 5.45 : SDS Blue Gene iş izi, HeteroSim yürütmesi (YBH metrikleri).



Şekil 5.46 : SDS Blue Gene iş izi, HeteroSim yürütmesi (AET metrikleri).

Bir diğer iş izi çalıştırma denemesi ise Ohio Supercomputer Center (OSC) Linux Custer'dan elde edilmiş iş iziyle yapılmıştır. 32 adet x 4 çekirdekli Pentium III Xeon işlemcili (550MHz, 2GB bellek) makine ve 25 adet x iki çekirdekli (550 MHz) makinadan oluşan Cluster'da 80.712 adet job yürütmesinin sonuçları toplu halde Şekil 5.47 ve Şekil 5.48'de gösterilmiştir. Benzer denemeler başka iş yükü izleri için de yapılmış ve HeteroSim ile sorunsuz olarak yüzbin satırın üstündeki iş yükü dosyalarının oynatılabildiği görülmüştür.



Şekil 5.47 : OSC Cluster, HeteroSim yürütmesi sonuçları.

6. KAVRAM İSPATI ÇALIŞMASI-1: TARBİL İÇİN PBM

Bu bölümde, tezde geliştirilen araçların gerçek sistemlerle kullanılması için yürütülen kavram ispat çalışmalarından ilki açıklanmıştır. Bu amaçla yüksek başarımlı bir sistem örneği olan, İTÜ TARBİL (Tarımsal İzleme ve Bilgi Sistemi) Projesi bilişim altyapısında, seçilen bileşenlerin politika tabanlı yönetilmesi konusu irdelenmiştir. Bu PoC, iki aşamalı olarak yürütülmüştür. İlk aşamada, TARBİL sistemi altyapısının bir YBH yapısına (bulut formunda) dönüştürülmesi yönünde çalışmalar sürdürülmüş (ANALİZ ÇALIŞMASI-1.1), ikinci aşamada ise TARBİL MSVD alt sisteminin YBH modeline dönüştürülmesi hedeflenmiştir (ANALİZ ÇALIŞMASI-1.2).

6.1 TARBİL Sistemin Tanıtımı

Sahadaki yer istasyonlarından ölçülen çeşitli veriler, uydu görüntüleri ile birlikte kullanılarak, parsel çözünürlüğünde, karar destek, verim tahmini, alan tahmini ve kayıp tahmini çıkartılmaktadır. Bu sistemde, analizler yapan uygulama sayısı gün geçtikçe artmaktadır. İTÜ TARBİL Sistemi IT altyapısının 400 çekirdekli mimarisi, incelenerek Şekil 6.1'deki gibi özetlenmiştir. Sistem, YBH sistemlerinde kullanılmaya müsait olan heterojen bileşenlerle (IBM, Dell sunucular, IBM depolama makinaları) kurulmuştur. Sistem, sensörlerden (7/24) sürekli ölçüm değerleri toplar ve depolar. Uzak kullanıcılar, web sunucu üzerinden sisteme bağlanıp interaktif olarak sistemde çeşitli sorgular yapmakta, çeşitli bilgileri yönetmektedirler. Uydu görüntüleme ile elde edilen fotoğraflar ve diğer coğrafi bilgiler, gelişmiş GPU özellikli DELL sunucular üzerine analiz edilmektedir. Sahadan toplanan sensör verilerini depolayan ve üzerlerinde analizler yapan uygulamalar ise IBM x3650 sunucular üzerinde ve birbirinden ayrıık olarak çalışmaktadır. x3650 sunucular, özellikle BüyükVeri ve Bulut uygulamaları için geliştirilmiştir [201].

Bütün işletim sistemlerinde koşan uygulamalar, yüksek kapasiteli *IBM Storwize V7000* depolama ortamında kendileri için oluşturulmuş depolama alanlarını kullanır. Storwize V7000, dosya ve blok veri depolamayı tek bir sistemde birleştirerek depolama sanallaştırması sağlayan, yüksek bandgenişlikli depolama sistemidir.

Uzun vadede, artan sensör sayısı, artan kullanıcı sayısı ve toplanan veriler üzerinde analizler yapacak yazılımların artan sayısı ve karmaşıklığı düşünüldüğünde, sistemin performansını da maksimize etme ihtiyacı olabileceği değerlendirilmektedir. Bu amaçla, sistem kaynaklarını uygulamalara daha etkin olarak paylaştıran bir RMS altyapısının kurulması gerekecektir. Bulut mimarisi, sistemin heterojen bileşenleri ve interaktif uygulamalar düşünüldüğünde uygun seçenek olarak öne çıkmaktadır.

6.2 Analiz Çalışması-1.1 : TARBİL Sisteminin Politikalarla Yönetilebilir YBH Yapısına Dönüştürülmesi

ANALİZ ÇALIŞMASI-1.1 kapsamında, TARBİL sistem bileşenlerinin özellikleri incelenerek, hem sistemin yönetiminin etkinliğinin artırılması hem de tez kapsamında geliştirilen teknolojilerin bu sistemin yönetiminde nasıl kullanılabileceği irdelenmiştir.

Sistem, donanım altyapısının idamesi, kullanıcıların kaynaklara erişim denetimi, ağ yönetimi, yazılım servislerinin donanım kaynaklarını verimli kullanabilmesi gibi çeşitli konularda yönetim ihtiyacı duyar. Bu ihtiyaçların çoğu, çeşitli yazılım araçları kullanılarak operatörlerce karşılanmaktadır. Sistemde, donanım altyapısının önemli kısmını oluşturan IBM ve DELL sunucuların yönetimi, üretici firmaların yönetim uygulamaları ile zaten yapılabilmektedir. Ayrıca, yönetmek için özelleşmiş ticari araçlar da bulunmaktadır. Donanım, yönlendirici, *switch*, depolama ortamları, ve veritabanlarının vb. yönetilmesi için ayrı ayrı yönetim yazılımları bulunabilmektedir (Örneğin IBM Tivoli yönetim araçları ailesi ve IBM Platform LSF).

Bu alternatiflerin çoğu, donanım bileşenlerinin alt seviyede parametrelerinin yönetilmesine odaklanır. Yönetim ihtiyacını, bileşen bazında ayrı ayrı karşılamaya çalışmak, (artan bileşen çeşitliliği nedeniyle) birbiri ile entegre olmayan, bir çok yönetim yazılımı ve yönetim adacıkları oluşmasına yol açar. Bölüm 3.1’de de açıklandığı üzere insan gücü yoğun ve hataya açık olan bu yaklaşım yerine Politika Tabanlı Yönetim sistemleri, yönetim ihtiyaçlarını bütünleşik olarak karşılayabilir. Ancak, TARBİL gibi büyük ve çok çeşitli ticari ürün içeren bir sistem için PBM altyapısı kurmak ve bütün bileşenleri tümleşik olarak yönetmek, kolay bir problem değildir. Zaten ticari bileşenlerin standart olmayan arayüzleri buna izin vermeyecektir.

Bu tür bir yüksek başarılı sistemde kaynakların kullanıcılara tahsis için Bölüm 2’de açıklanan çeşitli mimariler ve araçlar kullanılabilir. Ancak, kaynak tahsisleri ve altyapının yönetimini birbirinden ayrı düşünmek, bütün sistem bileşenlerinin tümleşik yönetimi ile elde edilecek faydaları sağlayamaz.

TARBİL gibi dağıtık sistemlerin yönetimi için, sanallaştırma ve bulut bilişim teknolojileri gittikçe daha çok kullanılmaktadır. Dolayısıyla TARBİL sistemi kaynaklarının işletilmesinde bulut bilişim altyapılarından birisinin kullanımına doğru gidilmesi, değerlendirilmesi gereken bir alternatiftir.

OpenStack, RackStack, CloudStack, Eucalyptus, Ganeti ve AppScale gibi bulut bilişim platformları bulunmaktadır. İzleyen alt bölümde, TARBİL sistemi bileşenleri ile de uyumlu olan, IBM ve DELL firmalarının desteklediği ve bağımsız değerlendirmelerde rakiplerine göre üstün özellikleri ile öne çıkan OpenStack bulut altyapısı detaylı olarak incelenmiştir.

TARBİL sisteminde bu tür bir kullanım şemasına geçilmesi ve OpenStack ile uygulamalar/kullanıcılar tarafından kaynak kullanımının sağlanması, açık kaynak kodlu olması sayesinde PBM’in entegre edilerek kullanılabilmesini sağlayabilecektir. Zaten bu servisler, zengin RESTAPI’ler sundukları için bu servislerle etkileşen programlar geliştirmek kolaydır.

6.2.1 Analiz çalışması-1.1 yol haritası

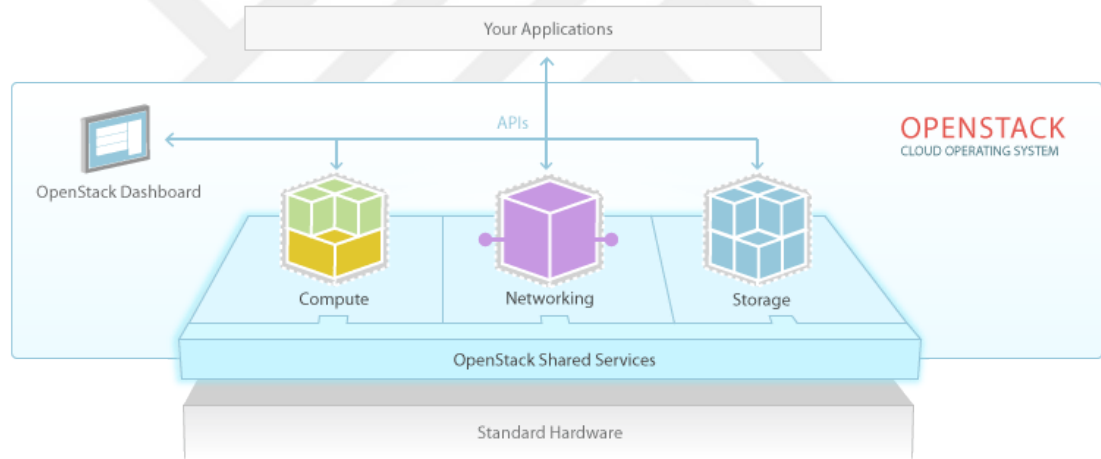
Sistemin politika tabanlı olarak yönetilebilir olmasını sağlayacak çalışmalar, kısa ve uzun vadeli olarak aşağıdaki gibi sınıflanmıştır:

- Kısa vade çalışmaları:
 - POLICE platformunun PEP bileşeni ile OpenStack Bulut servisleri entegrasyonu için prototip gerçekleştirilmesi yapılması. Bu entegrasyon için Apache jclouds (SDK) kütüphanesi kullanılmaktadır. *jclouds*, bulut uygulamaları geliştirmek için tasarlanmış Java tabanlı açık kaynak araç kümesidir (<http://jclouds.apache.org>).
 - DevStack (test amaçlı mini OpenStack sürümü), TryStack ve gerçek OpenStack kurulumlarında test edilmesi
 - Keystone, Ceilometer OpenStack servislerinin modifikasyonu ile tam PEP entegrasyonunun sağlanması

- TARBİL altyapısında tahsis edilecek test amaçlı makinalarda OpenStack kurulumlu deneme.
- Uzun vade çalışmaları:
 - Bulut yönetim politikalarının belirlenmesi ve POLICE ile yapılabilirliğinin sağlanması
 - Horizon GUI servisi içine POLICE arayüzlerinin eklenmesi
 - TARBİL sisteminde OpenStack bulut sistemine geçiş (IBM ve DELL sunucular)

6.2.2 OpenStack bulut platformu

Bu bölümde, OpenStack teknolojisi ve TARBİL bileşenlerinin bu platformla uyumluluğu irdelenmiştir. Ayrıca, OpenStack platformunun politika tabanlı yönetimi için mevcut ve geliştirilmekte olan teknolojiler irdelenerek, kendi PBM platformumuzun entegrasyonu için elde edilen fikirler de bu bölümde verilmiştir.



Şekil 6.2 : OpenStack servis çeşitleri.

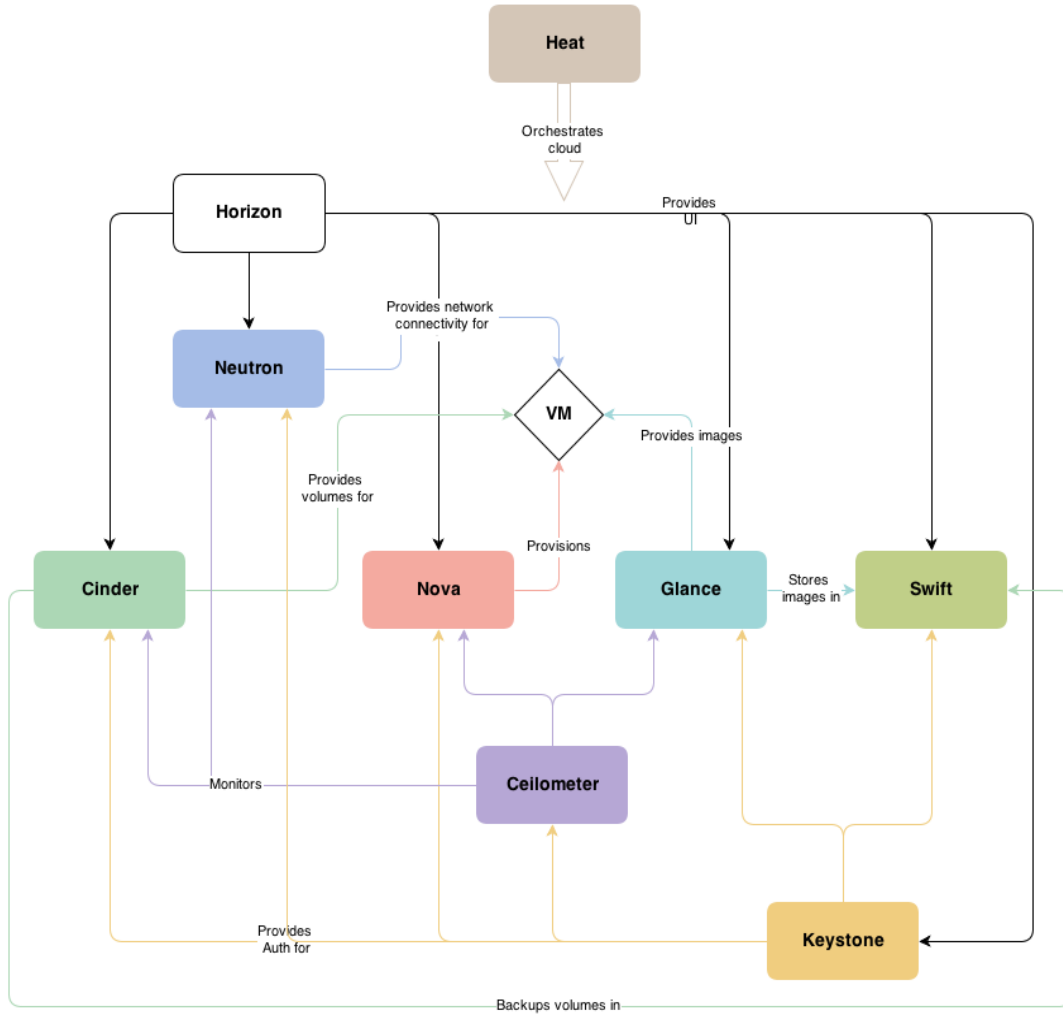
OpenStack, açık kaynak kodlu kurumsal ölçekte bulut hesaplama işletim sistemi yazılımıdır [202]. Bir *IaaS (Infrastructure as a Service)* çözümü olarak kullanılır. Openstack, biribiri ile ilişkili ama farklı geliştirme takımlarınca yürütülen bir dizi projeler halinde geliştirilmektedir (Şekil 6.2).

OpenStack IBM geliştirmeleri, işyükü gönderme, sanal imaj yönetimi ve gözleme için bir portalin yanında gelişmiş sıralama bileşeni sağlar. Otomasyon, ölçme ve güvenlik diğer özelliklerdir. OpenStack tabanlı bulut sistemi kurmak ve işletmek, yoğun teknolojik bilgi gerektirir. Çünkü OpenStack, çok esnektir ve çok fazla konfigüre

edilebilir seçenek sunar. Yanlış konfigürasyon, sistemde kararsızlık ve güvenlik riski oluşmasına neden olabilir.

6.2.2.1 OpenStack servisleri

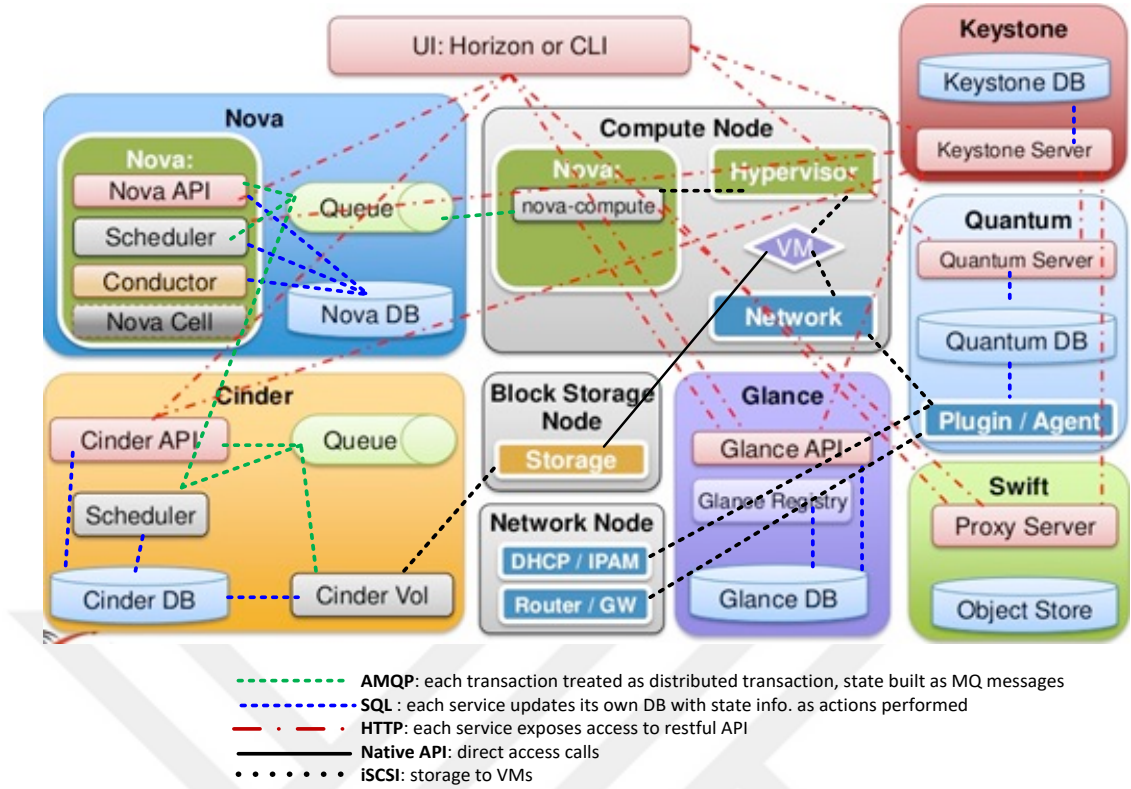
OpenStack bazı ortak servisler içerir. *Identity*, imaj yönetimi ve web arayüzü gibi servisler, OpenStack bileşenlerini birbirine ve dış sistemlere bağlarlar (Şekil 6.3).



Şekil 6.3 : Juno sürümü OpenStack servisleri etkileşimi.

Her servisin kendine özel veritabanı ve veri depolama katmanı vardır. Görevlerini yerine getirirken birbirleri ile RESTAPI arayüzleri üzerinden etkileşimleri gerekir (Şekil 6.4).

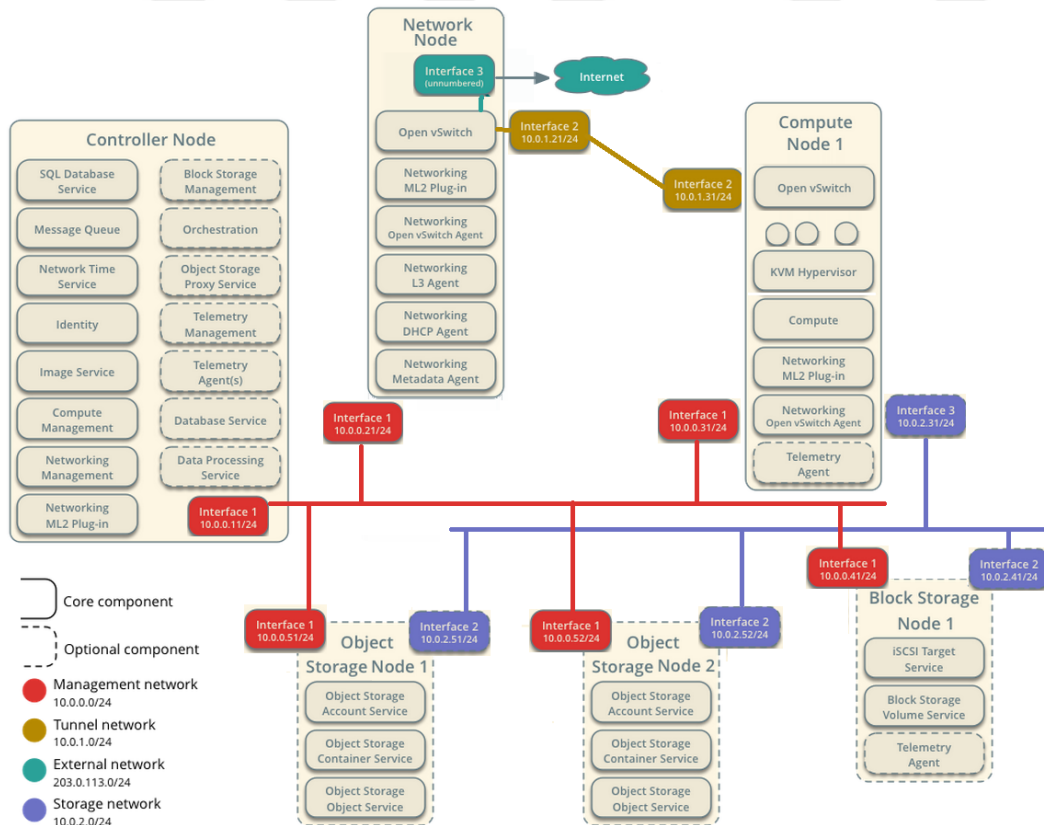
OpenStack servislerinin kısa tanımları aşağıdaki listede verilmiştir ve Şekil 6.5’de ise senaryolar ile örneklendirilmiştir:



Şekil 6.4 : OpenStack servisleri ve iletişim mekanizmaları.

- **Kimlik Servisi (Keystone):** OpenStack servislerine erişen kullanıcıların merkezi kimlik doğrulama servisidir. Ayrıca, bulutta sunulan bütün servislerin merkezi kayıt servisi olarak da hizmet verir. Sistem yöneticileri, kullanıcılar ve sistemlerle ilgili merkezi politikaları ayarlayabilirler.
- **Imaj Servisi (Glance):** Disk ve Sunucu imajlarının kaydettirilmesi, aranıp bulunması, paylaşılması, mevcut sunucu imajlarının kopyalanması, canlı kopyasının oluşturulması, kaydedilmiş imajdan yeni sunucular oluşturulması, yedeklerinin alınması hizmetleri sunar.
- **Telemetry servisi (Ceilometer):** Bulutta çalışan servislerin performans ve kullanım bilgilerini toplar. Servisler için faturalandırma servisidir. Bulut çapında veya kaynaklar çözünürlüğünde metriklerin görülebilmesini sağlar.
- **Orchestration servisi (Heat):** Hazır şemaları kullanarak uygulama geliştiricilerin altyapıların, servis ve uygulamalar dahil konuşlandırılmasını tanımlama ve otomatize edebilmesini sağlar.

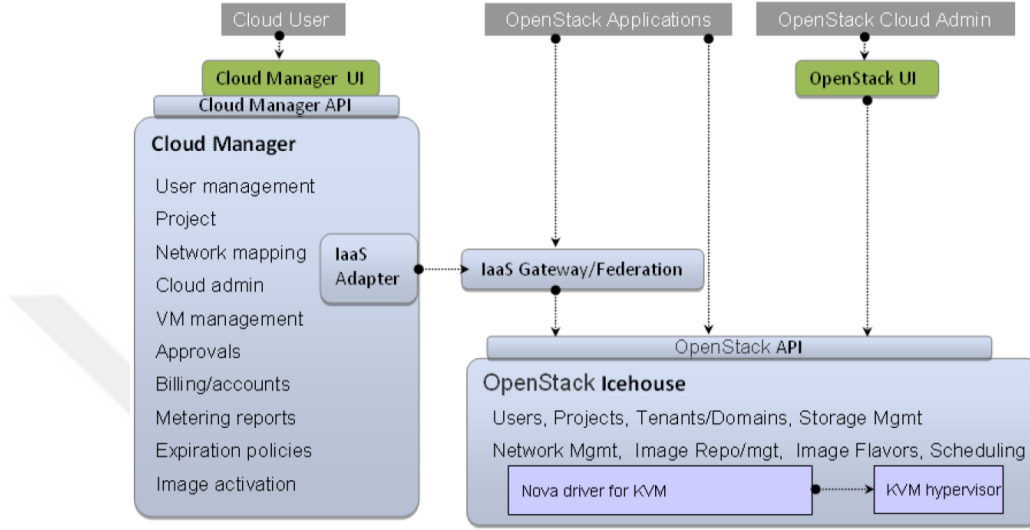
- Veritabanı Servisi (Trove): Kullanıcıların detaylarla (yönetim, konfigürasyon, yedekleme, yedekten kurtarma, izleme, yama yükleme vb.) uğraşmadan, hızlıca veritabanı kullanabilmesini sağlar.
- Sanal Ağ Servisi (Neutron, önceki adı Quantum): Tenant'lar için birbirinden izole, sanal şebekeler oluşturmaya izin verir.
- Compute Servisi (Nova): Geniş sanal makine şebekeleri sağlar, yönetir.
- Object Storage Capabilities (Swift): Ölçeklenebilir (petabytelerca) veri depolama yeteneğidir. Statik veriler (sanal makine imajları, fotoğraf, e-posta depoları, arşivler, yedekler) için dağıtık depolama sunar.
- Block Storage Capabilities (Cinder): Hesaplama düğümlerine blok seviyesinde depolama cihazları sağlar. Performans gerektiren senaryolara uygundur. VM Kopyalarının kendi içlerindeki veriler kalıcı değildir. Kopya ile birlikte yok olurlar. Bu servisi ise Sanal makine kopyalarına kalıcı depolama sunar.
- Dashboard (Horizon): Kaynakların ve servislerin yönetilebilmesi için Web arayüzü sağlayan servistir.



Şekil 6.5 : Örnek OpenStack konuşlandırma görünümü (yeniden çizim).

6.2.2.2 IBM bulut desteği

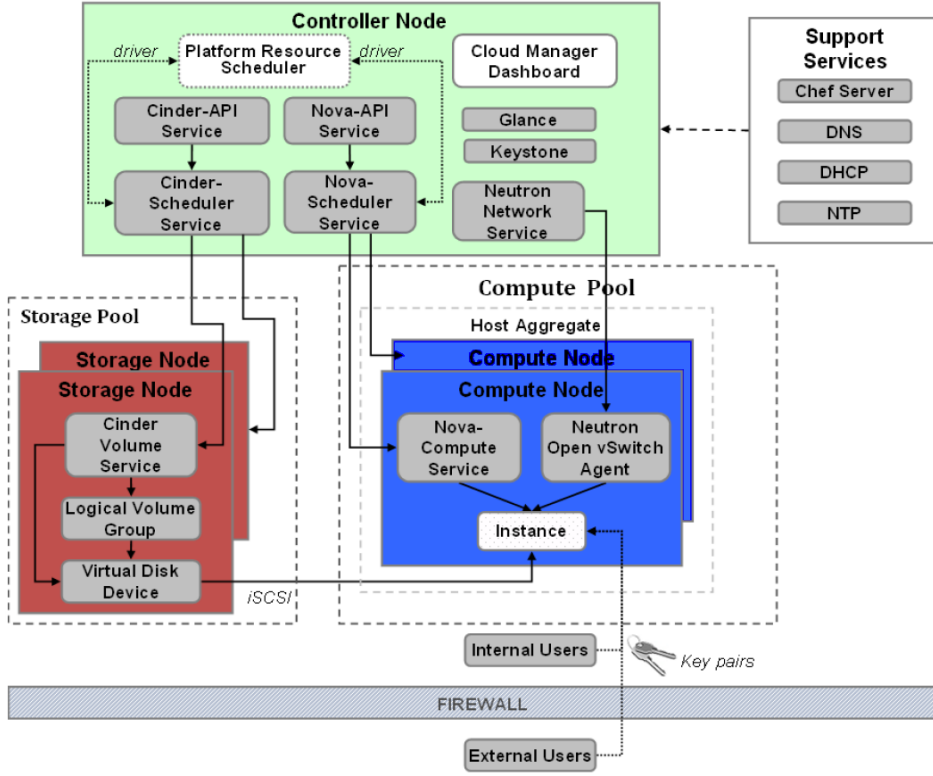
IBM, OpenStack platformunun, kendi donanım ürünlerinin yönetiminde kullanılabilmesini sağlamak için *IBM Cloud Manager with OpenStack* isimli bulut yönetim yazılımı sunmaktadır (Şekil 6.6). Bu bölümde verilen bilgiler, IBM referans dokümanlarından derlenmiştir [203-207].



Şekil 6.6 : IBM Cloud Manager mimarisi [203].

IBM Cloud Manager with OpenStack bir IaaS geçit üzerinden OpenStack ile entegre olur. *Cloud Manager UI* kullanıcıların bulut altyapısını işletmesini sağlar. Bu modelin Şekil 6.7’de gösterilen bileşenleri (OpenStack komutlarının, kaynak tahsislerini denetleme, ağ konfigürasyonlarının yapılması, faturalandırma, ölçüm, istek ve onay akışı desteği vb.) aşağıdakilerdir:

- **Tenant (Project):** OpenStack sistemlerinde, kaynakları/kullanıcıları birbirinden izole etmek veya gruplamak için kullanılan kavramdır. İzole kaynakları, ayrı ağları, depolama alanları, imajları, kullanıcıları olan tahsis tanımlarıdır. Kaynaklara kota sınırları tenant çözünürlüğünde yapılabilir.
- **AvailabilityZone:** Kullanıcının yeni kaynakları tanımladıkları yerlere yerleştirmesine olanak sağlar. İki tipi vardır (instance: VMs ve volume:VirtualBlockStorageDevices)
- **Host Aggregate:** Availability Zone’u daha alt parçalara böler. Daha gelişmiş iş planlaması yapabilmek için makine gruplarına Key-Value çiftleri atanabilir.
- **Flavor:** Sunucular için, RAM boyu, vCPU, ağ bandgenişliği ve disk alanı kombinasyonu ile belirlenen sınıfları ifade eder.



Şekil 6.7 : IBM Cloud Manager bileşen modeli [203].

- Compute pool: Nova ile sanallaştırılmış hesaplama sunucuları içerir. *IBM Cloud Manager with OpenStack*, IBM platformlarında sanallaştırma sağlar.
- Storage pool: Birçok depolama sunucusu içerir. Bu sunucular, kendi yerel disk sürücülerini üzerine kalıcı blok storage kaynakları sağlarlar.
- Support servisleri: OpenStack bulut ortamındaki ana servisleri desteklemek için DNS, DHCP veya Network Time Protocol (NTP) servisleri çalıştıran çevresel servislerdir.
- Controller: Kullanıcılara bulut kaynak sağlama amaçlı işlem yapma olanağı sunan portalı içeren merkezi yönetim sunucusudur. Hesaplama ve depolama havuzlarında çalışan servislere erişim ve zaman planlaması yapar.
- Platform Resource Scheduler: PRS, OpenStack kaynaklarının kullanım planması için kullanılır. VM kopyalarının hangi düğümlere konuşlandırılacağı, taşınacağı, optimizasyonu gibi kararları vermede kullanılır. Konuşlandırma politikaları kullanılır. Politika (aslında strateji kavramı daha doğru bir tanımlamadır) tipleri şunlardır: İş yüklerini en az fiziksel kaynağa topla, Arıza toleransı için iş yüklerini mümkün olduğunca çok fiziksel sunucuya dağıt, Yeni iş yüklerini, düşük yüklü fiziksel sunuculara ata vb.

Platformun sağladığı esneklikler nedeniyle OpenStack ortamı konuşlandırması için çok çeşitli modeller vardır. IBM, kendi System x donanımlarında OpenStack konuşlandırmasının konfigürasyonu için bir referans mimari de yayımlamıştır. Bu yapı, Red Hat Linux 6.5 yüklü olan IBM System x3650 M4, IBM System x3550 M4 ve IBM System x3650 M4 BD sunucularını *IBM Cloud Manager with OpenStack 4.1* yazılım kümesi ile kullanır. Ağ konfigürasyonu, IBM System Networking RackSwitch G8124E ve G8052 içerir. IBM platformları olan TARBİL makinalarının OpenStack ile kullanımına geçiş için gerekli sürücü yazılımları mevcuttur.

6.2.2.3 OpenStack platformunun politika tabanlı yönetimi

OpenStack bulut platformunun, yönetiminin de basitleştirilmesi gerekmektedir. Bu amaçla politikalarla veya ön tanımlı konfigürasyonlarla yönetimi basitleştirmeyi hedefleyen çeşitli çalışmalar sürmektedir. İzleyen bölümde tanıtılan, *Congress* Projesi ve *Purview(Tetris)* Projesi gibi çalışmalar sürse de OpenStack için IETF modelinde bir PBM çözümü henüz bulunmamaktadır.

A. Mevcut Openstack sürümündeki yönetilebilirlik yetenekleri

Mevcut OpenStack gerçekleymesinde, basit ama dağınık bir politika desteği bulunur. Her OpenStack servisi, kaynak yönetim kuralları içeren, JSON formatında bir politika dosyasına sahiptir (policy.json). Kurallarla, API erişimi, depolama alanı kullanımı, sanal makine kopyası başlatma vb. eylemler yönetilir. Her servisin kendi politika uygulamasının bulunması, sistemde dağınık bir görüntü oluşturur. Bazı servislerin politikaları değiştirilebilirken bazılarının ise sabittir. Politikaların, servislerin tamamını mimarisi ve durumu ile uyumluluğunu sağlayacak bir merkezi akıl yoktur. Dahası Openstack modüllerini geliştiren proje gruplarının bazılarında, politika kavramı olması gerekenden farklı anlamlarda kullanılmaktadır. Örneğin *swift* projesinde politika kavramı, *swift.conf* dosyasına yazılan ve sabit değeri olan, depolama çalışma modu konfigürasyonlarıdır (iki kopya, üç kopya, parçalı sakla). Sanal Ağ Servisi (Neutron) için sürdürülmekte olan politika çalışması, gerçek anlamdaki PBM hedeflerine daha yakındır. *<classifier, action>* çiftleri şeklinde kurallar tanımlanması hedeflenmektedir. *Action* kısmı şimdilik, *allow*, *deny*, *drop* değerlerinden birisini almaktadır; *qos*, *redirect* gibi eylemlerin de eklenmesi planlanmaktadır. Ancak, sadece önceden tanımlı ve politikaların desteklediği eylem tipleri kullanılabilir. Örnek bir kural tanımlama isteği aşağıda verilmiştir.

POST v1.0/policy_rules.json

Content-Type: application/json

Accept: application/json{

```
"name": "web-rule",  
"direction": "ingress",  
"port_range_min": 80,  
"port_range_max": 80,  
"protocol": "tcp"  
"ethertype": "IPv4"
```

}

Block Storage servisi *policy.jsonfile* dosyası örneği aşağıda verilmiştir. Dosyaya göre, *default rule*, kullanıcının ya *admin* ya da depolama alanının sahibi olması gerektiğini belirtir. Sadece sahibi veya admin, oluşturma, silme, güncelleme yapabilirken, disk alanı tipleri yönetimi gibi diğer bazı işlemleri sadece *admin* yapabilmektedir.

```
{  
  "context_is_admin": "role:admin",  
  "admin_or_owner": "is_admin:True or project_id:%(project_id)s",  
  "default": "rule:admin_or_owner",  
  "admin_api": "is_admin:True",  
  "volume:create": "",  
  "volume:get_all": "",  
  "volume:get_volume_metadata": "",  
  "volume:get_volume_admin_metadata": "rule:admin_api",  
  "volume:delete_volume_admin_metadata": "rule:admin_api",  
  "volume:update_volume_admin_metadata": "rule:admin_api",  
  "volume:get_snapshot": "",  
  "volume:get_all_snapshots": "",  
  "volume:extend": "",  
  "volume:update_readonly_flag": "",  
  "volume:retype": "",  
  "volume_extension:types_manage": "rule:admin_api",  
  "volume_extension:types_extra_specs": "rule:admin_api",  
  "volume_extension:volume_type_encryption": "rule:admin_api",  
  "volume_extension:volume_encryption_metadata": "rule:admin_or_owner",  
  "volume_extension:extended_snapshot_attributes": "",  
  "volume_extension:volume_image_metadata": "",  
  "volume_extension:quotas:show": "",  
  ...  
  "volume_extension:volume_host_attribute": "rule:admin_api",  
  "volume_extension:volume_tenant_attribute": "rule:admin_or_owner",  
  "volume_extension:volume_mig_status_attribute": "rule:admin_api",  
  "volume_extension:hosts": "rule:admin_api",  
  "volume_extension:services": "rule:admin_api",  
  "volume_extension:volume_manage": "rule:admin_api",  
  "volume_extension:volume_unmanage": "rule:admin_api",  
  ...  
  "snapshot_extension:snapshot_actions:update_snapshot_status": "",  
}
```

```

"consistencygroup:create" : "group:nobody",
"consistencygroup:delete": "group:nobody",
"consistencygroup:get": "group:nobody",
"consistencygroup:get_all": "group:nobody",
"consistencygroup:create_cgsnapshot" : "",
"consistencygroup:delete_cgsnapshot": "",
"consistencygroup:get_cgsnapshot": "",
"consistencygroup:get_all_cgsnapshots": "",
"scheduler_extension:scheduler_stats:get_pools" : "rule:admin_api"
}

```

Politikalar, bulut yöneticisi tarafından tanımlanıp güncellenebilir. OpenStack iki tip yetki politikası barındırır:

- Operasyon tabanlı: Politikalar, operasyonlara özel erişim kriterleri belirler.
- Kaynak tabanlı: Kayağa erişim için yetki tanımlanabilir

OpenStack servisleri için politika uygulanması, servisten servise değişkenlik gösterebilir. Policy engine, *policy.json* dosyasını okur. Dosyaların yeri sürümünden sürüme değişkenlik gösterebilir (nova servisinde */etc/nova/policy.json*). Sistem çalışırken dosya güncellenebilir. Yeniden başlatma gerekmeden devreye alınabilir. Örnek politika aşağıdaki gibidir:

```

compute:create:[["rule:admin_or_owner"]] ifadesinde,
politika → compute:create ve rule → admin_or_owner.

```

Politikalar, bir OpenStack API operasyonu veya operasyonda kullanılan bir öznitelik ile eşleştğinde, OpenStack Policy Engine (OPE) tarafından otomatik olarak tetiklenir. Yetki politikası bir veya daha fazla kural içerebilir. Bu kuralların herhangi birisi başarılı değerlendirilirse politika başarılı değerlendirilmiş olur. Eğer, bir API operasyonu birden fazla politika ile eşleşirse, bütün politikaların başarılı olarak değerlendirmeyi geçmesi gerekir. Yetki kuralları rekürsiftir. Sonlandırıcı kural bulununcaya kadar içiçe değerlendirme akış sürer. Tanımlanan kurallar:

- Rol tabanlı kurallar: İsteği gönderen kullanıcının rolü uygun ise kural başarılı değerlendirilir. Örnek: "role:admin".
- Özellik tabanlı kurallar: İstekte belirtilen kaynağın özelliği belirlenmiş değerde ise başarılı kuraldır. Örnek: "field:networks:shared=True" ağ kaynağının paylaşım özneliği *true* değerinde ise kural başarılı.
- Genel kurallar: Kaynağın bir özneliğindeki değer ile kullanıcının güvenlik parametrelerinden çıkarılan bir öznitelik değeri karşılaştırılır. Örnek:

"tenant_id:%(tenant_id)s" canlı kaynak kopyasının *tenant kimliği* ile isteği gönderen kullanıcının tenant kimliği eşit ise kural başarılıdır.

Örnek *nova* servisi *policy.jsonfile* parçası aşağıda verilmiştir:

```
{
  "context_is_admin": [["role:admin"]],
  "admin_or_owner": [["is_admin:True"], ["project_id:%(project_id)s"]],
  "default": [["rule:admin_or_owner"]],
  "compute:create": [ ],
  "compute:create:attach_network": [ ],
  "compute:create:attach_volume": [ ],
  "compute:get_all": [ ],
  "admin_api": [["is_admin:True"]],
  "compute_extension:accounts": [["rule:admin_api"]],
  "compute_extension:admin_actions": [["rule:admin_api"]],
  "compute_extension:admin_actions:pause": [["rule:admin_or_owner"]],
  "compute_extension:admin_actions:unpause": [["rule:admin_or_owner"]],
  ...
  "compute_extension:admin_actions:migrate": [["rule:admin_api"]],
  "compute_extension:aggregates": [["rule:admin_api"]],
  "compute_extension:certificates": [ ],
  ...
  "compute_extension:flavorextraspecs": [ ],
  "compute_extension:flavormanage": [["rule:admin_api"]],
}
```

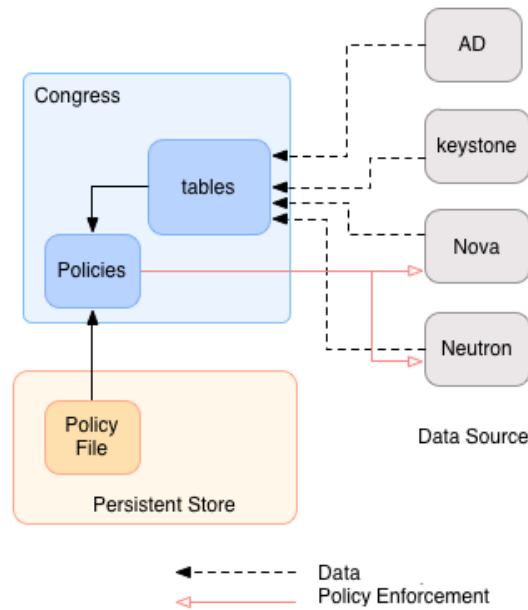
Yukarıdaki kuralda eğer aktif kullanıcı admin rolünde ise veya istekte belirtilen kaynağın sahibi ise (tenant kimliği eşit ise) kural başarılı değerlendirilir. Örnekte ayrıca, fiziksel kaynakları (flavor) manipüle etme yetkisi Admin API'yi kullanan admin tipli kullanıcılarla sınırlandırılmıştır. Eğer örnekteki dosyaya aşağıdaki satır eklense idi kullanıcılara kendi sunucu kaynaklarını (flavor) oluşturma yetkisi de verilmiş olurdu: *"compute_extension:flavormanage": []*.

B. Geliştirilmekte olan PBM yetenekleri

Congress isimli OpenStack projesinde, bütün bulut servisleri genelindeki dinamik değişikliklerle uyumlu bir yönetim yeteneğini, *policy as a service* olarak sunmak için çalışmalar sürdürülmektedir. Aşağıdaki fonksiyonellikler hedeflenmektedir:

- Üst seviye ve genel amaçlı bir tanımlama dili kullanılarak iş mantığının tanımlanması. Politika dili sabit politika tipleri ve politika uygulama mekanizmaları içermez. Bunun yerine, politikalarda, servislerce Congress'e sağlanan durum bilgilerinden, bulutun hangi durumlarının uygun olduğu hangilerinin uygun olmadığı tanımlanır. Örneğin:
 - Uygulama-A sadece Uygulama-B ile haberleşmeye izinlidir.
 - Tenant-A'nın sanal makinası, eğer Tenant-A bir B grubunun üyesi ise her zaman bir açık ağ bağlantısına sahip olmalıdır.
 - Sanal Makine-A, Depolama-B kaynağından farklı bir coğrafi bölgeye konulmamalıdır.
- Herhangi bir bulut servisinin kolayca bağlanabileceği modüler-bağlanılabilirlik sağlamak
- Politikaları uygulamak
 - Politika çakışmalarını olmadan önlemek
 - Çakışmalar olduktan sonra düzeltmek
 - Yöneticilere, politikalar ve çakışmaları (ne olduklarını ve nasıl düzeltilebileceklerini) tanımlama olanağı sunmak.

Politika tipleri arasında modal ve anlamsal çakışma tesbit ve çözme yetenekleri henüz yoktur. Politika uygulama yapısı (Şekil 6.8), *Yasaklı Eylemler*, *Yasaklı Durumlar* ve *Yürütülecek Eylemler* isimli üç tip tabloda tutulan bilgilere dayanır.



Şekil 6.8 : Congress politika uygulama bileşen diyagramı.

Kullanılan politika dili, Prolog türevi olan Datalog'dur. Bu dil, geleneksel programlama dili söz dizimine benzeyen SQL benzeri bir dildir. Politikaları tablolar kümesi olarak tanımlar. Dilin grameri aşağıdaki gibidir:

```

<policy> ::= <rule>*
<rule> ::= <atom> COLONMINUS <literal> (COMMA <literal>)*
<literal> ::= <atom>
<literal> ::= NOT <atom>
<atom> ::= TABLENAME LPAREN <term> (COMMA <term>)* RPAREN
<term> ::= INTEGER | FLOAT | STRING | VARIABLE

```

Buna göre bir kural *HEAD:BODY* formatındadır. *IF BODY THEN HEAD* şeklinde okunur. Congress politikalar için bir dizi tablo kullanır. *error* tablosundaki satır, bir politika ile olan ihlali gösterir. Her kuralın BODY kısmı bir dizi (tabloda bulunmayanları göstermek üzere negatif de olabilen) tablonun AND ile birleşmesinden oluşur; HEAD kısmı tek tablodur. Negatif olamaz.

- : Negatif
 , : AND
 OR : Aynı tablo adlı iki HEAD

Kuralın formatı ve bir örneği aşağıda verilmiştir:

$table_0(kolon1, kolon2, \dots) :- [-]table_1(kolon1, \dots), \dots, [-]table_m(kolon1, \dots)$

Politika: Bir VM'e bağlı her ağ, ya açık ya da özel olmalı ve VM'in sahibi ile aynı grupta yer alan bir sahibi olmalı.

İhlal: A kullanıcısının VM'i, B kullanıcısına ait ağa bağlı ama A ve B ortak bir grupta değil.

error :- nova:vm(vm),	
neutron:network(network),	Burada not same-group(vm-own, net-
nova:network(vm, network),	own)
not neutron:public(network),	ldap::group(user-id,group-name)
neutron:private(network),	tablosunu kullanarak aynı gruba üyeliği
nova:owner(vm, vm-own),	tanımlar. Aynı grupta olan kullanıcılar
neutron:owner(network, net-own)	için <i>same-group</i> tablosu oluşur.

Şekil 6.9 : Congress (Classification) politikası ve ihlal durumu örneği.

Congress'in kullandığı politika tipleri aşağıda açıklanmıştır:

- Classification policy: Politikaları ihlal eden bulut durumları ve sebepleri tanımlanır (Örnek: yukarıdaki *error* politikası).

- **Enforcement Policy:** Bulutun durumuna göre Congress'in hangi eylemi yürüteceğini tanımlar. Kural, eylemin yürütüleceği koşulu tanımlar. Örnek ihlalcı bir VM tesbit edildiğinde *disconnectNetwork* eylemi çalıştırılmalıdır:
`execute(disconnectNetwork(vm, network)) :-`
`nova:vm(vm), neutron:network(network), nova:network(vm, network),`
`-neutron:public(network), neutron:private(network),`
`nova:owner(vm, vm-own), neutron:owner(network, net-own),`
`-same-group(vm-own, net-own)`
- **Access Control policy:** Hangi bulut koşullarında, hangi kullanıcının, hangi eylemleri yürütmeye yetkili olduğunu tanımlar. Örnek: *disconnectNetwork* eylemini yürütme yetkisi, sadece VM'in sahibi veya admine verilir.
`permit(disconnectNetwork(vm, network), user) :- nova:owner(vm, owner),`
`-equal(user,owner), -ldap:group(user, "admin")`
- **Action Description policy:** Congress tarafından yürütülebilir eylemleri ve bulut kaynakları üzerindeki etkilerini tanımlar. Örnek, *disconnectNetwork(vm, network)* eyleminin bulut servis tabloları üzerindeki etkilerini tanımlayan kural: (*disconnectNetwork(vm,network)* eylemi, *nova:network* tablosundan *<vm, network>* satırını siler. Eylemi ancak VM sahibi yürütebilir).
`nova:network-(vm, network) :-`
`disconnectNetwork(vm, network),`
`nova:owner(vm, owner),`
`username(owner)`

Ayrıca sadece HEAD kısmı olan kural eylemler tablosu da tanımlanabilir. Politikaların merkezi tek noktadan tanımlanabilmesi, daha tutarlı yönetim sağlar.

Bir diğer önemli çalışma ise *Purview (Tetris)* projesidir. Congress projesi ile birleşme kararı almıştır. Farklı tipte politika uygulayabilen bir PBM geliştirmeyi hedefler. Politika örnekleri aşağıda verilmiştir:

- **Availability Policies**
 - Düğümün çökmekte olduğuna dair işaretler (bazı donanım alarmları) varsa VM'ler başka düğümlere taşı.
 - Düğüm çökerse veya hypervisor yazılım yükseltmesi yapılacaksa, bütün VM'leri başka bir düğümde yeniden başlat.

- Toplam enerji tüketimini minimize etmek için sistem yük durumuna göre düğümleri aç/kapat. Yüğü düşük olan CPU'lardaki VM'leri bir araya toplayıp boşalan düğümleri kapat. Yüğü miktarı artınca da kapalı düğümlerden (hypervisors) açarak dengelemeye çalış.
- Performance Policies
 - Düğüğü kullanım oranı 100%'e yaklaşırsa bazı VM'leri diğery düğümlere taşıyarak VCPU/PCPU oranını yükselt.
- Load Balancing Policy
 - Bazı CPU'lar aşırı yüğüli diğeryleri düşük yüğüli ise bazı VM'leri düğümler arasında taşıyarak sistemi dengele
- User Defined Policy
 - Operatörler bazı özel metrikler tanımlayıp bunların değerylerine göre bazı politikalar tanımlayabilirler.

Politika formatı, ACTIONS'ları tetikleyen CONDITIONS şeklindedir. Eylemleri tetikleyecek ifadeler, operatörler tarafından kolayca anlaşılabilir RULES veya ALGORITHMS olarak belirtilebilmektedir. Politikalar arası çakışma kotarma desteğı yoktur. Modellerinde tasarladıkları politika otomasyonu bileşenleri şunlardır:

- Conditions
 - VM_FAIL gibi sabit ifadeler içerir. Koşul ifadesi, nova üzerinden VM durumlarını kontrol eden bir python class örneğinde yapılacaktır.
 - Koşul ifadeleri, parametrik olabilecektir.
- Actions
 - VM_RESTART gibi jenerik ifadeler içerir.
 - Eylem ifadeleri, parametrik olabilecektir.
- Policy definitions: Politikalar bir dizin servisinde tutulacaktır. Politikalar, OpenStack API çağrıları detayını gizleyen soyutlukta olacaktır.
 - Örnek: { if VM_FAIL(vm1) then VM_RESTART(vm1) }
 - Politikalar rules veya algorithms şeklinde tanımlanabilecektir.
 - Politika tanımını bir YAML² dosyasına yazılabilecek.
- Parser: politika dosyasını yorumlayıp eylem ve koşulları veritabanına yazar.

² 2001 yılında Clark Evans tarafından önerilen, üst seviye programlama dilleri tarafından kolayca kullanılabilir formatta veriler üretilebilen rekürsif veri serileştirme şeması

- PAE (Policy Automation Engine): Koşulları sürekli izleyip eylemleri tetikler.
- API Server: Politikalar üzerine manipülasyon için restful API sağlar:
 - Politika listeleme, pasifleştirme/aktifleştirme, durumunu görme (kaç kez ve ne zaman tetiklendi vb.)
- Client: API Server ile etkileşim için istemci kütüphane ve komut satırı desteği.

Örnek bileşenler Şekil 6.10’da verilmiştir.

<pre> file actions/vm_actions.py class VMigrateAction(ActionBase): """Migrate a Virtual Machine :param vm_name: virtual machine name :param node_name: name of the compute node the VM will migrate to """ def action(self, context, vm_name=None, node_name): class VMRestartAction(ActionBase): """Restart a Virtual Machine :param vm_name: virtual machine name """ def action(self, context, vm_name): </pre>	<pre> file actions/compute_node_actions.py class NodeStopDeployAction(ActionBase): """Stop deploying to this Node :param node_name: compute node name """ class NodeRebootAction(ActionBase): """Reboot a node :param node_name: compute node name class NodeReprovisionAction(ActionBase): """Reprovision a node :param node_name: compute node name </pre>
<pre> file setup.cfg [entry_points] actions = vm.migrate = tetris.actions.vm_actions:VMigrateAction vm.restart = tetris.actions.vm_actions:VMRestartAction node.stopdeploy = tetris.actions.compute_node_actions:NodeStopDeployAction node.reboot = tetris.actions.compute_node_actions:NodeRebootAction node.reprovision = tetris.actions.compute_node_actions:NodeReprovisionAction </pre>	

Şekil 6.10 : Tetris politika otomasyon bileşen örnekleri.

Koşullar, Eylemlere benzer. Yeni bir eylem tipi, yapılacak işlemi içeren yeni bir python gerçekleştirme sınıfı demektir. Örneğin, *vm.fail* ‘e ait python sınıfında (VMFail) nova python API çağrısı yapılarak bir VM’in durumu kontrol edilir. Her koşulun *evaluate* metodu, koşulu sağlayan varlıkları içeren key/value listesi döndürür. Örnek, NodeutilizationThreshold koşulunun *evaluate (node_name=None, threshold=0.5)* metodunun çıktısı:

```
[ { 'id': "node1", 'utilization': 0.54}, { 'id': "node2", 'utilization': 0.75}, ]
```

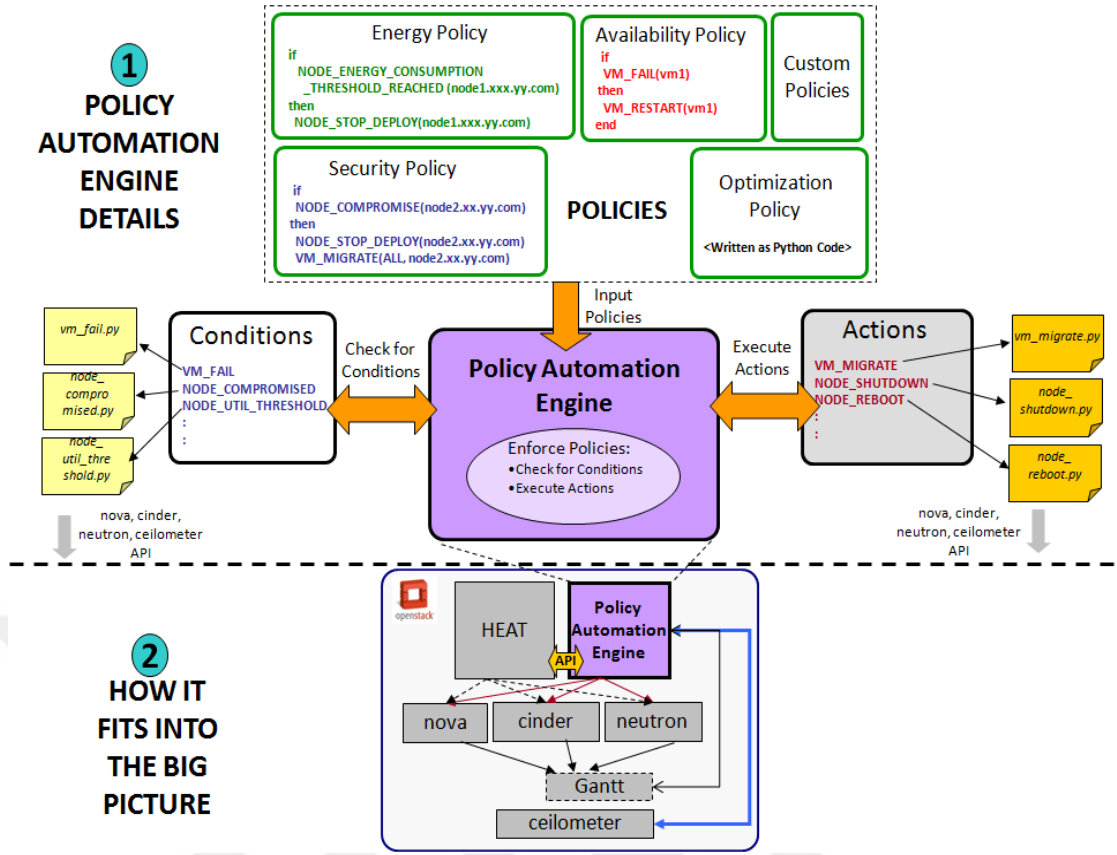
Üç adet politika kuralı içeren YAML formatında dosya örneği Çizelge 6.1’de verilmiştir:

Çizelge 6.1 : Kullanıcı açısından metrikler (N adet iş yükü için)³.

availability policy snippet	energy policy snippet	security policy snippet
<pre>- name: Security enabled: 1 interval: 3600 stabalization: 5 conditions: - name: "node.compromise" parameters: node_name: "node2.xx.yy.com" actions: - name: "node.stopdelpoy" parameters: node_name: "node2.xx.yy.com" - name: "vm.migrate" parameters: node_name: "node3.xx.yy.com" user_supplied_policy1: policy1_parameters: </pre>	<pre>- name: Availability enabled: 1 interval: 300 stabalization: 5 conditions: - name: "vm.fail" parameters: vm_name: "vm1" actions: - name: "vm.restart" parameters: vm_name: "vm1"</pre>	<pre>- name: Energy enabled: 0 interval: 10800 stabalization: 5 conditions: -name: "node.energy.threshold" parameters: node_name: None threashold: 90 actions: - name: "node.stopdeploy" parameters: node_name: "\$id"</pre>

Her policy plugin, kendi YAML dosyasını yorumlayıp yürütme için gereken PAE görevlerini oluşturmaktan sorumludur. Tetris mimarisi ve gerçekleştirme görünümü, Tetris proje sitesindeki tasarım dosyasından alınan Şekil 6.11’de gösterilmiştir.

- ³ name: politikanın adı;
- enabled: kural aktif mi? Ön tanımlı değeri false;
- interval: koşul kısmının değerlendirme periyodu (saniye);
- stabalization: eylemleri yürütmeden önce koşul ifadesinin kaç kez true dönmüş olması gerekir.



Şekil 6.11 : Tetris gerçekleştirme görünümü.

Koşullar ve eylemler python kodu olarak ifade edilip politikalar içinde anahtar kelimeler ile adreslenebilir (Örnek: Availability Policy). Koşul ve eylemler içinde python kodları da çağrılabilir. PAE, sunucu süreci olarak gerçekleşir. Koşulları sürekli izler. Politikalarla ilgili çağrılar (listeleme, geçmişi görme, ihlaller vb.) cevaplar. Ceilometer servisi üzerinden metrikleri öğrenir, alarmlara kayıt olup dinler. Eylem ve koşulları kodlayanlarla politikaları tanımlayanların aynı kişiler olması gerekmez. Tasarladıkları politika dilleri çok basittir. Ayrıca, tek bir PAE ile uygulanacak politika sayısı da çok artabilir.

6.2.3 POLICE ile OpenStack yönetimi entegrasyonu

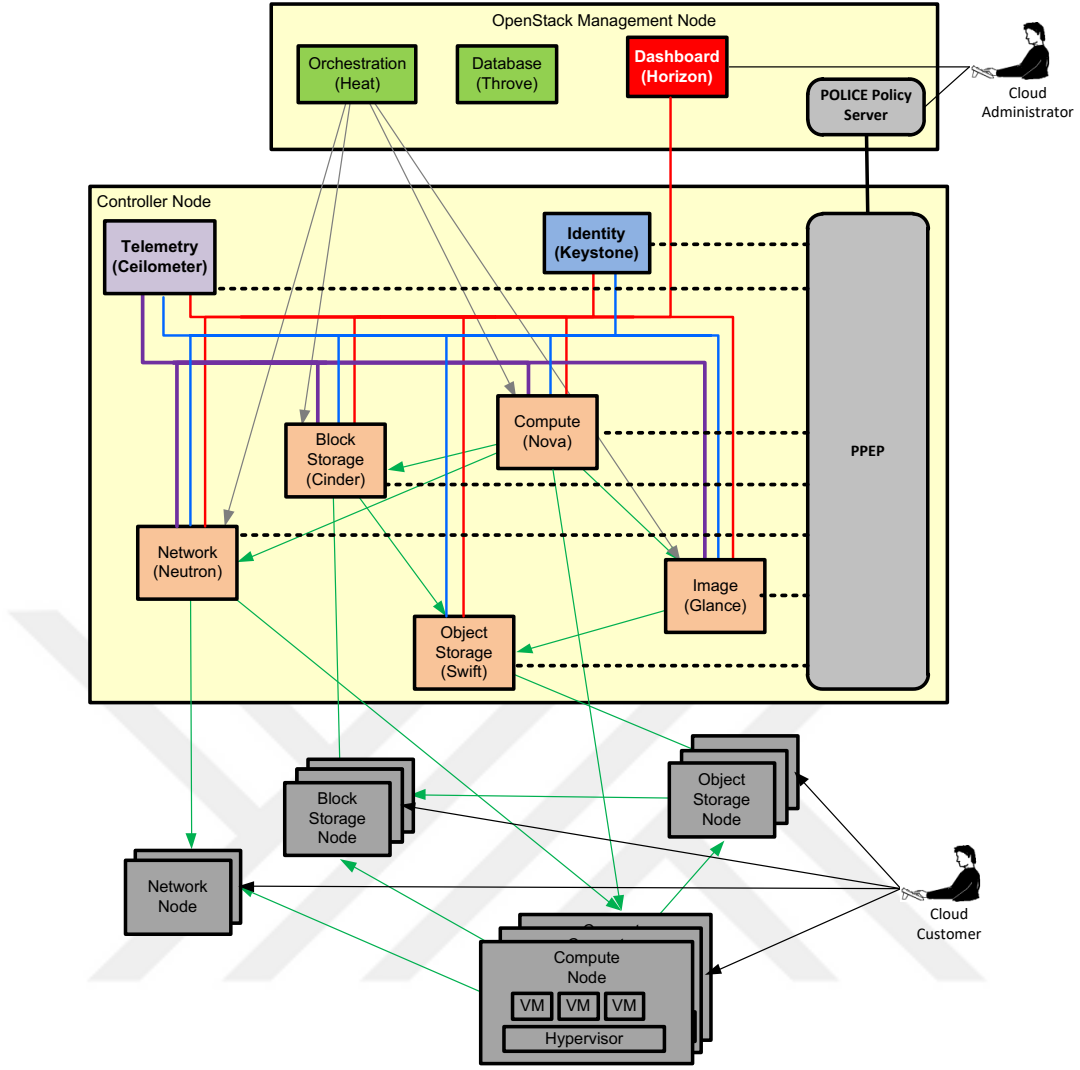
Öncelikle OpenStack platformunda politikaları uygulamak için sürdürülen Tetris ve Congress projelerindeki yaklaşım alternatifleri incelenmiştir:

- Proaktif politika uygulama: Politika ihlallerinin olmadan önleendiği ideal seçenektir. Bunu yapabilmek için OpenStack servislerinin her türlü eylemlerinden önce PEP'e sorgu göndermesi gerekir. Gerçekleme maliyeti en yüksek olan seçenektir. VM ağ bağlantı değişimlerinin, Grup üyelik değişimlerinin, VM sahiplik değişikliklerinin PEP emrinde olması demektir.

- Reaktif politika uygulama: Politika ihlali oluřtuęunda (Bulut gereklemesi proaktif uygulamaya izin vermedięi iin ihlallerin nlemedięi varsayımıyla) dzeltici eylemlerin yapılmasıdır. Dzeltici eylem iin yeterli bilgi bulunduęu ve dzeltici eylemleri yrtebildięi varsayılır.
- İnteraktif politika uygulama: sttekiler gibi otomatik deęildir ve ynetici kullanıcıya kararını sorarak onun kararına gre eylemi uygulamaktır.
- Yardımlı politika uygulama: Servisler, bulutun ve politikaların mevcut durumu ile baęlantılı/uyumlu bilgilerle alıřırlar. Bylece, sistem durumu politikalar ile uyumlu řekilde (ihlal sayısını artırmadan) deęiřebilir. rnek politika iin, Nova compute servisi, VM'i konfigre eden operatre ihlalleri artırmayacak aęların listesini gsterek yardım edebilir.

Bu tanımlardan anlařıldıęı gibi sistemde oluřabilecek ihlal durumlarını nlemek iin ncelikle yardımlı/proaktif uygulama kullanılmalı; mmkn deęilse de reaktif uygulama kullanılmalıdır. Congress modeli, bulutun davranıřını ve politikalarla ne kadar uyumlu alıřtıęını izler. Farklılıklar belirlerse politika ihlali olarak iřaretler. Congress, bařlangıta Nova, Neutron ve LDAP servislerini sorgulayıp gncel bilgilerini ęrenir. Bazı servisler iin srekli yoklama gerekirken bazı servisler gncelleme gnderir. Congress, bulut servislerini ihlal tesbiti iin izleme, ihlal dzeltme ve nleme iin eylemleri yrtme tr iřler yapar.

POLICE, OpenStack'in ihtiya duyduęu politika tiplerini tanımlayabilmeye uygundur. OpenStack ile tam entegrasyon saęlanarak geliřtirilmekte olan dięer PBM zmlerinin aksine, yukarıdaki uygulama seeneklerinden Proaktif politika uygulamayı yapabilir. OpenStack ile POLICE atısının entegrasyonu iin tasarlanan senaryonun btn Cloud mimarisi iindeki yeri řekil 6.12'de gsterilmiřtir. Bu diyagramda hem OpenStack servislerinin birbiri ile etkileřim yolları hem de POLICE bileřenlerinin bu bileřenlerle etkileřimi gsterilmiřtir.



Şekil 6.12 : POLICE ile OpenStack PBM senaryosu.

Şekil 6.13, OpenStack istemci kullanıcı veya uygulamalarının sistem servislerini kullanma senaryosunun detayını gösterir. Bu detaydan görülebileceği üzere Keystone



Şekil 6.13 : Mevcut OpenStack token tabanlı erişim denetimi senaryosu.

servisi PEP kullanılarak kontrol altında tutulursa servislerle kullanıcıların etkileşimi de kolayca yönetilebilir. Bununla birlikte, servislerin birbiri ile etkileşimi, sistemde olan

olaylar karşısında yürütülmesi istenen eylemlerin de PBM kapsamına alınabilmesi için PEP ile diğer servislerin entegrasyonunun sağlanması gerekir. Politika uygulama için, Bölüm 3.2.3.4’da tanıtilan PEP – Yönetilen Sistem arasındaki yönetim arayüzü soyutlamasında (NMI) yönetilen nesneler (MO) ile ilgili aşağıdaki fonksiyonları kullanılabilir. Olay servisinin, sisitemdeki olayları izlemesi için *Ceilometer* servisi kullanılabilir. NMI fonksiyonlarının OpenStack servisleri ile entegre edilmesi gerekir. *Ceilometer* servisi kullanılarak tek bir alarm veya birbirini tetikleyecek alarm zinciri tanımlaması yapıp dinlenebilmektedir. Alarmları tanımlarken, tanımlanan bir metrik verilip belirlenen değerlere ulaştığında alarmin tetiklenmesi sağlanabilir.

Etki alanları (Domain), kullanıcı, grup ve proje kümelerini temsil eder. Her biri sadece bir etki alanına aittir. Kullanıcılar birden fazla proje (başka etki alanından bile olabilir) ile ilişkilendirilebilir.

OpenStack içinde yönetilecek belli başlı MO (ManagedObject)’ler ComputeNode, NetworkNode, StorageNode, User, Admin, xService, VM’lerdir.

Elde edilen mimaride, PEP *as a Service* (PEPaaS) yaklaşımına uygun olarak diğer bütün servislere yönetim hizmeti sunulmuş olacaktır.

6.3 Analiz Çalışması-1.2: TARBİL Sistemi MSVD Uygulamasının Politikalarla Yönetilebilir Şekilde Paralleştirilmesi

Bütün TARBİL sistem altyapısını Bulut sistemine geçirme çalışması (POC-1 ANALİZ ÇALIŞMASI-1.1), tez takvimi içerisinde tamamlanması riskli olduğundan, sistemin alt kümesinin yönetim kapsamına sokulması alternatifi seçilmiştir. Kapsam, TARBİL sistemindeki MSVD uygulamasını içerecek şekilde değiştirilmiştir. Böylece, POLICE PBM aracının YBH sistemlerinin yönetiminde kullanılabilirliğini deneysel olarak uygulama ile göstermek için TARBİL sistemindeki MSVD bileşenlerini politika tabanlı yönetilebilir hale getirmek amaçlanmıştır. Bu amaç için aşağıdaki adımların izlenmesi planlanmıştır:

- Mevcut MSVD sistemi bileşenlerini, YBH mimarisi bileşenlerine çevirmek
- YBH altyapısı kurmak
- YBH altyapısını politika tabanlı yönetmek
- İyileşmeleri incelemek

Çalışmada tamamlanan adımlar aşağıdakilerdir:

- Mevcut TARBİL yazılım/donanım yapısı (çok çekirdekli altyapıyı etkin kullanmaya çalışan özel yazılım gerçekleştirilmesi) incelenip modellenmiştir
- Mevcut sistem süreçleri için kaba ölçümler toplanmıştır
- Süreçler için YBH yapısına uygun atomik İş Parçacıkları belirlenmiştir
- TARBİL sisteminde YBH gerçekleştirilmesi için kullanılabilecek alternatifler arasında İş Parçacığı Yönetim Sistemi (JMS) taraması yapıldı. Çalışma-2 için bir JMS platformu seçimi yapılmıştır
- JMS sistemi ile POLICE PEP entegrasyonu kodlaması yapılmıştır.
- MSVD uygulamasının JMS üzerinde çalışır hale getirilmesi için kodlama çalışması başlatılmış ama tamamlanmamıştır.

6.3.1 TARBİL süreç yapısı ve iş yükü tipi sınıflandırması

TARBİL Sisteminde MSV (Mobil Sistem Veri Sunumu) kapsamında, çeşitli veritabanlarından alınan veriler derlenip, çeşitli sunum formatlarında sistem kullanıcılarına ve diğer uygulamalara sunulur. MSV iki katmandan oluşur: Veri Derleme (MSVD) ve Veri Sunumu (MSVI). MSV Servisi, zirai, meteorolojik bilgileri, uyarıları ve haritaları oluşturup web sunucu üzerinden sunar. MSVD (Mobil Sistem Veri Derleme), sahadaki sensörlerden alınan verileri istatistiksel ve coğrafi analizlerden geçirerek kronolojik veri kümelerini üretip sistemin kullanıcılarına sunan alt sistemdir. Farklı makinalar üzerinde çalışan parçalardan oluşur.

- MSVD Hesaplayıcı Bilgisayarları: Sistem operatörlerine, iş yüklerini (noktasal ölçüm verilerinden coğrafi haritaların üretilmesi) istedikleri makinada manuel olarak çalıştırma imkanı sunulur. Operatörler, makinaların yük durumunu kontrol edip uygun bir makinada yeni bir işlemi başlatırlar. İş yükü, IDWEC (*Inverse Distance Weighting with Elevation Correction*) sayısal yükseklik modeli ile düzeltilmiş mesafe ters orantılı interpolasyon tekniği kullanılarak yapılır. 3D saha modeli ve noktasal ölçüm değerleri kullanılarak sahanın tamamındaki değerler, interpolasyon ile hesaplanır ve harita formatına getirilir.
- MSVD Kayıtcı Bilgisayarları: Kendilerine (MSVD Hesapçılar tarafından) gönderilen veriler içerisinde, kendi paylarına düşen (önceden konfigüre edilmiş) parçayı alıp işleyip diske kaydederler. Kayıtcıların, verileri diskteki dosyalarına yazmaları sırasında zaman zaman darboğaz oluşmaktadır.

MSVD yazılımının yüksek başarımlı olarak çalıştırılması amacına, halihazırda, birden fazla bilgisayar üzerinde manuel olarak çalıştırılan benzer işlemler ile ulaşılmaktadır. Sistemde, gelen iş yükü isteğini, bilgisayarlara otomatik olarak dağıtan gerçek anlamda bir YBH yeteneği bulunmamaktadır.

Sahadaki sensör verileri 10 dakikalık periyotlarla web sunucuya kaydedtirilir. Ham ölçüm verisi olarak *ÖlçümDB* bilgisayarında depolanan bu veriler, hem MSV tarafından kullanılır hem de web sunucu tarafından dış dünyaya sunulur. *ÖlçümDB*'deki ham veriler üzerindeki bir sonraki işlem, *VerifiedDB* makinasındaki *DATA_TAMAMLAMA* yazılımı tarafından yürütülür. Bu 10 dakikalık ham sensör verileri alınarak öncelikle eksik ölçümler düzeltilir. Daha sonra bu verilerden, 10 dakikalık, 30 dakikalık, günlük ve aylık periyotlu istatistiksel (sensör tipine göre maksimum, minimum, ortalama, toplam vb.) veriler hesaplanarak veritabanına kaydedilir. Hesaplanan bu veriler, hem web sunucu üzerinden dış dünyaya sunulur hem de MSVD kayıtçıların bir sonraki hesaplamasına girdi olarak kullanılır.

MSVD Hesaplayıcı makinalarındaki *INTERPOLASYON HESAPLAMA ARAYÜZÜ (IHA)* adlı yazılım çalıştırıldığında, doğrulanmış ölçüm verilerini *VerifiedDB* makinasından alır ve alana yayma hesaplaması yapar. *IHA* yazılımı, sürekli çalışan bir servis uygulaması olarak planlanmasına rağmen operatörler tarafından manuel olarak her hesaplama için tekrar tekrar çalıştırılır. *IHA* yazılımının hesapladığı alana yayılmış veriler, depolanmak üzere MSVD Kayıtçı bilgisayarlarına yayınlanır. MSVD Kayıtçı Bilgisayarlarında servis olarak çalışan *DATA_SERVICE* yazılımları, yayını dinleyerek sadece kendileri ile ilgili olan veri parçasını alıp depolarlar. Yani, yayınlanan veri, 5 farklı bilgisayar üzerinde eş zamanlı olarak çalışan yazılım bileşenleri tarafından alınıp paralel olarak işlenip depolanmış olur.

IHA yazılımının diğer bir çıktısı olan alana yayılmış değerlerle üretilmiş olan haritalar, *VerifiedDB-HaritaKayıtçısı* makinasındaki *MAP_HOST* yazılımı tarafından depolanır ve dış dünyaya sunulur.

MSVD Kayıtçı makinasında manuel olarak tetiklenen Hesaplama Yazılımı, alana yayılmış değerlerden, günlük/aylık değerler hesaplar.

NDVI⁴ alt sistemi, uydu fotoğraflarının Türkiye haritası üzerine, coğrafi projeksiyon kullanarak oturtulması ve bu haritalardaki piksel değerlerinin (bitki örtüsüne göre) hesaplanması işlerini yapar. NDVI altsisteminin işleyeceği uydu görüntüleri, İTÜ UHUZAM (Uydu Haberleşme Uzaktan Algılama Merkezi) sunucularından manuel olarak web sunucunun da kullandığı dizine FTP ile transfer edilir.

MSV sistemi bir YBH Sistemi olarak planlanmasına rağmen güçlü donanım altyapısı üzerinde çalışan dağıtık bir sistem olarak çalıştırılmaktadır. İki adet büyük boyutlu depolama sistemi (IBM ve Huawei Storage), diğer bilgisayarlarca kullanılır. Bu iki depolama sistemi, iki ayrı yönetim yazılımı ile ayrı ayrı yönetilmektedir.

Sistemde manuel olarak tetiklenmeyen, servis olarak 7/24 çalışan işlemler şunlardır:

- VerifiedDB makinasının veri tamamlama ve istatistiksel verileri hesaplaması
- MSVD Kayıtların interpolasyon verilerini depolaması
- NDVI Kayıtların NDVI verilerini depolaması
- Web sunucunun web servisleri üzerinden dışarıya sunduğu işlemler
 - İstasyon ölçüm verilerinin merkezde kaydedilmesi
 - Hesaplanan MSV haritalarının dış kullanıcılara sunulması
 - Hesaplanan NDVI haritalarının dış kullanıcılara sunulması
 - Uzak kullanıcıların sistemde yaptıkları sorgulama, listeleme işlemleri

Halen manuel olarak çalıştırıldığı halde gelecekte otomatik çalıştırılabilecek işlemler ise şunlardır:

- MSVD interpolasyon hesaplama (manuel tetiklenerek asenkron çalıştırılıyor)
- MSVD günlük (halen sadece manuel tetiklenerek asenkron çalıştırılıyor)
- NDVI hesaplama (halen sadece manuel tetiklenerek çalıştırılıyor)
- Harita dosyaları ve veritabanlarının yedek merkezde replikasyonu
- Veritabanları üzerinden gevşek bağlı iş akışlarının (Örnek *DATA_TAMAMLAMA* ile *IHA* yazılımları arası) API tabanlı iletişime çevrilmesi

MSVD sisteminin, bir YBH görevi olarak çalışacak, tekrarlanabilir alt parçalara dönüştürülmesi için, öncelikle, MSVD sisteminin yazılım bileşenleri ve arayüzleri

⁴ NDVI (Normalized Difference Vegetation Index) uzaktan algılama verilerinin, gözlenen hedef alan içinde canlı yeşil bitki içerip içermediğini analiz etmek üzere kullanılabilir.

belirlenmiştir (Şekil 6.14). Daha sonra bileşenlerin içindeki yazılım parçaları tesbit edilmiştir. Bu yazılım parçalarının MSVD bütünsel süreci içinde hangi alt adımlara karşı düştüğü ve diğer adımlarla önde gelme ilişkileri belirlenmiştir (Şekil 6.15).

6.3.1.1 Yük tiplerinin karakteristiğinin modellenmesi

Sistemde, hesaplama düğümleri (fiziksel bilgisayarlar), depolama düğümleri (IBM, Huawei Storage) ve şebeke band genişliği tipinden kaynaklar bulunmaktadır. Sistemdeki bu kaynakların kullanılmasını gerektirecek yük çeşitleri ve kullandıkları kaynaklar Çizelge 6.2’de listelenmiştir:

Çizelge 6.2 : MSVD yazılımları, verileri ve kullandıkları kaynaklar.

Yük tipi	Veri tipi	Tetikleyici	Kullanılan kaynaklar
UHUZAM Harita transferi Servisler	Web Sunucuya gelen istekler •Başvuru (dönemsel) •İstasyon verisi kaydetme •Sorgulama/listeleme	Operatör tarafından tetiklenir Dış kullanıcılar İstasyonlar tarafından tetiklenir Ölçüm verilerinin sorgulanması, listelenmesi	WebSunucu Web Sunucu ÖlçümDB IBMStorage Network BW VerifiedDB ÖlçümDB IBMStorage Network BW
Replikasyonlar	Ank-İst arasında uydu görüntülerinin manuel olarak FTP ile yedeklenmesi	Operatör tarafından tetiklenir	IBM Storage Network BW
MSVD Hesaplama	MSVD Hesaplamaları	İç kullanıcılar tarafından tetiklenir	MSVDHesapçı VerifiedDB IBMStorage Network BW
MSVD Kayıt	MSVD kaydı	MSVD hesapçı tarafından tetiklenir.	MSVDKayıtçı HuaweiStorage Network BW
NDVI Hesaplama	NDVI değer hesabı	Operatör tarafından	NDVI Hesapçı Web Sunucu
NDVI Kayıt	NDVI harita kaydı	NDVIHesapçı tarafından	NDVI Kayıt Network BW

Sistemde bu kaynakları bir bütün olarak izleyip görevlere tahsis edecek bir kaynak yönetim sistemi mevcut değildir. Bu görevlerin ürettiği yüklerin karakteristiklerinin ve sistem kaynaklarına olan etkilerinin belirlenmesi için bütün kaynaklar üzerindeki yüklerin ölçülmesi gerekmiştir. Mevcut sistemde bu türden bütünleşik ve merkezi bir

izleme mekanizması bulunmamaktadır. Bunun yerine aşağıdaki parçalı izleme mekanizmaları bulunmaktadır:

- Her bilgisayarın işletim sisteminin logları
- Her bilgisayarın işletim sistemi anlık performans metrikleri (süreç başına cpu payı, bellek, disk, network kullanımı)
- Web sunucunun web servislerine çağrı geçmişi
- Herhangi bir bilgisayarın TCP/IP adreslerine erişim geçmişi
- Depolama yönetim yazılımlarının logları

Sistemin kaynak (PE:CPU/GPU, BW, RAM, Storage) kullanım optimizasyonu yapabilir olması hedeflenmektedir. Tüm kaynakların en etkin kullanımı yerine, kaynakların, iş yüklerince optimum olarak kullanılabilmesi hedeflenir (Donanımsal koşullar yerine verilen donanımın göre başarımın oransal optimizasyonu). Günlük belirlenmiş sayıda kullanıcının isteklerini karşılarken donanımın göre başarımın oransal optimizasyonunu sağlamalıdır. Ama bu hedefleri uygulayacak ve gerçekleşip gerçekleşmediğini sistem genelinde bütünleşik olarak izleyip değerlendirebilecek bir mekanizma sistemde bulunmamaktadır. Mevcut sistemde performans izlemesi için ancak, yazılımların üzerinde çalıştıkları bilgisayarların işletim sistemleri tarafından, süreçler (process) bazında sağlanan (CPU doluluk oranı, Bellek kullanımı, şebeke kullanımı vb.) performans parametrelerinin gözle kontrolü yapılabilmektedir. Bu yöntemle sistemi izlemenin ise aşağıdaki sakıncaları vardır:

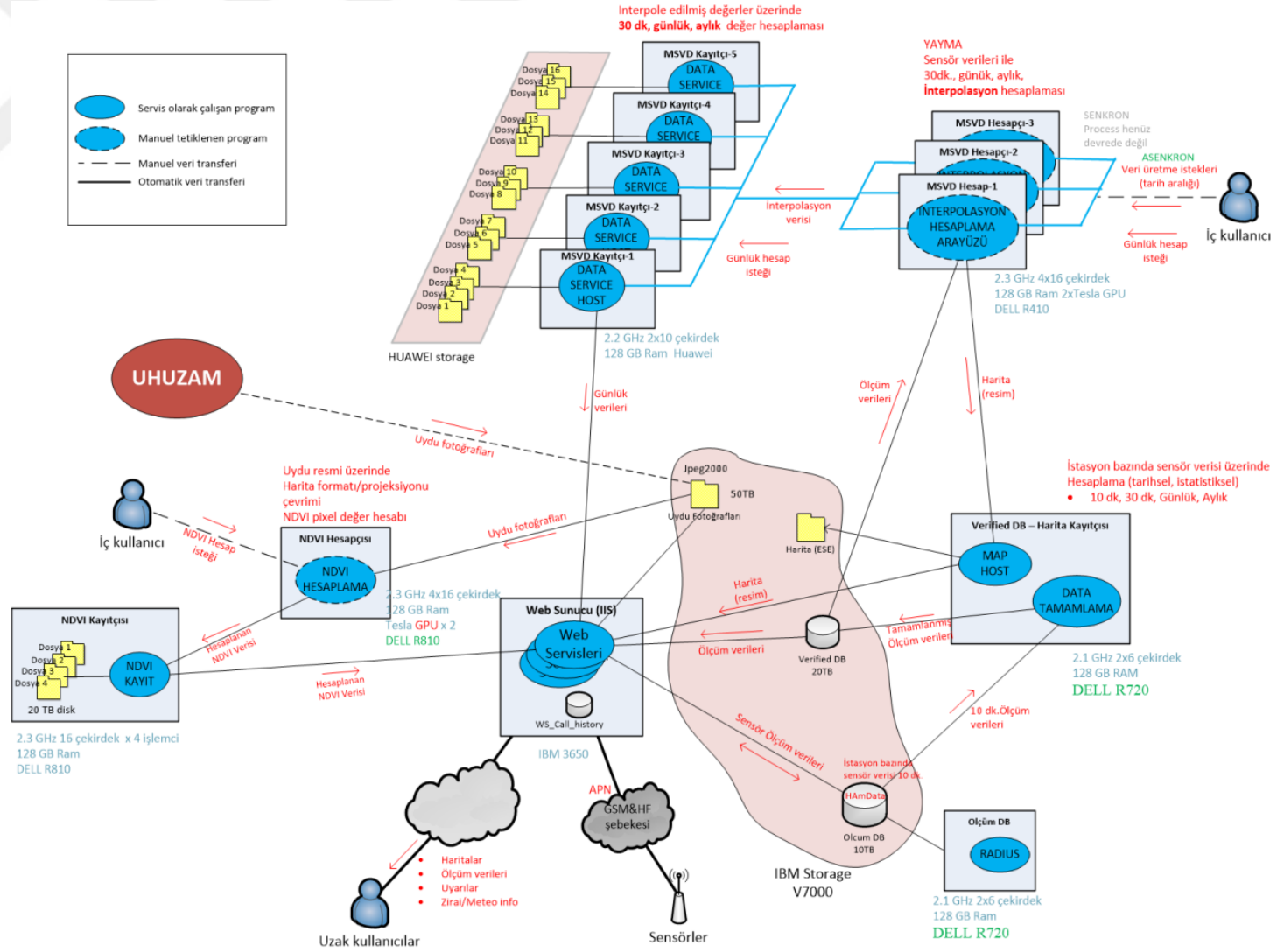
- Bilgisayarlarda ölçülen sürecin performans metriklerinin, gerçekte sistemden istenen hangi göreve/işleme denk düştüğü belirlenememektedir.
- Aynı bir isteğin yanıtlanmasında rol alan birden fazla/farklı bilgisayardaki performans parametrelerinin birbirleri ile olan korelasyonu hesaplanamamaktadır.
- Gözlenen değerler otomatik olan herhangi bir karar verme sürecinde kullanılmamaktadır.

Sistemin mevcut logları ve/veya eklenecek yeni izleme mekanizmalarına ait çıktıların incelenmesi sonucunda, kaynakları verimli kullanmayı engelleyen görev tipleri ve kaynak atama mekanizmaları belirlenebilecektir. Bununla birlikte bu inceleme öncesinde de bazı alt bileşenlerde iyileştirilebilecek konular değerlendirilebilir.

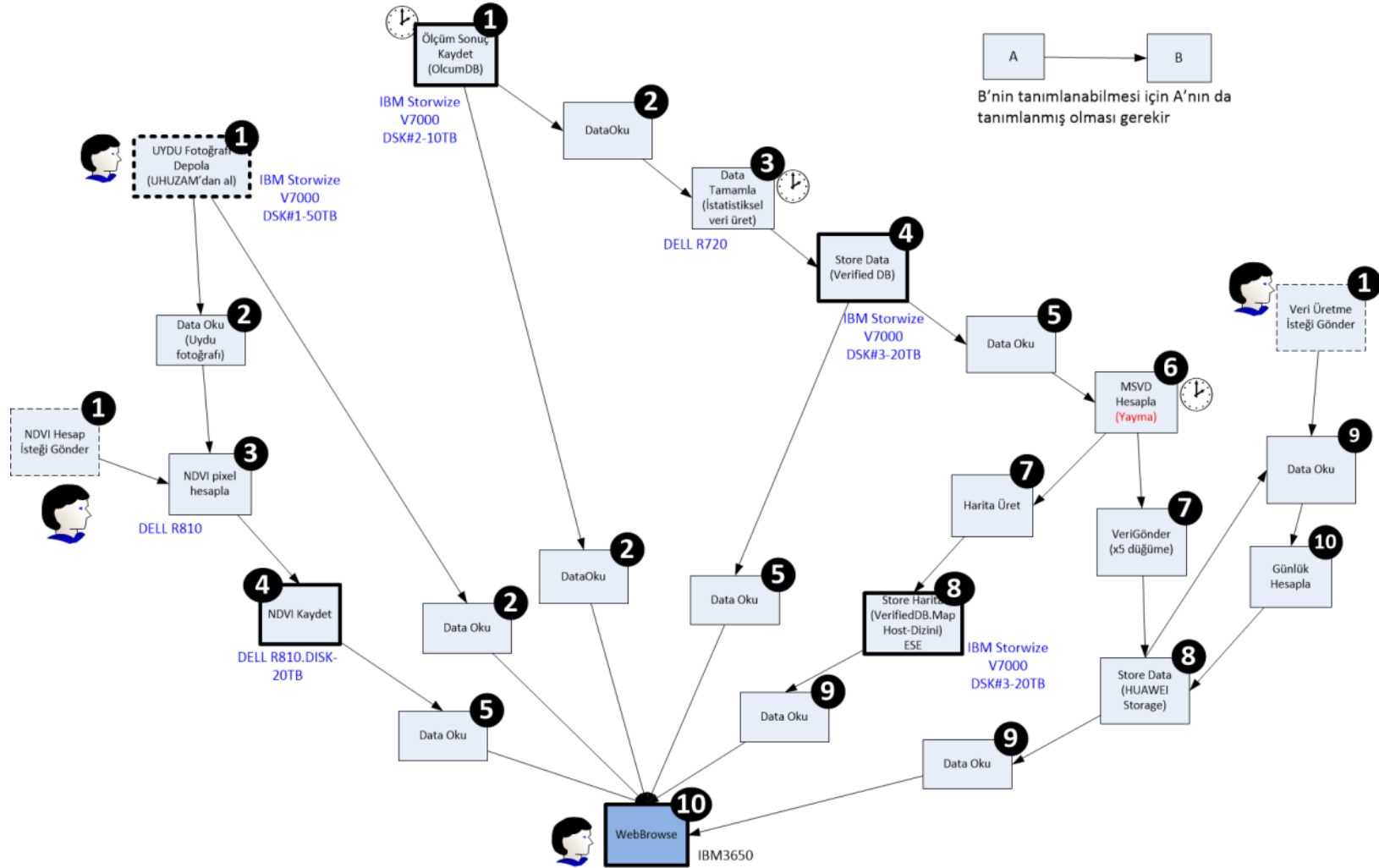
MSVD İş/kaynak (Workload/Resource) Yönetim mekanizmasının (RMS) kurulması için aşağıdaki adımların atılması gerekmektedir:

- Uygun İş/Kaynak Yönetim sisteminin seçilmesi ve PBM tabanlı yönetilebilir hale getirilmesi
 - SLURM, HTCondor, Torque vb. mevcut RMS'leri inceleyerek TARBİL sistemine uygun olanı belirlemek
- RMS'in mevcut MSVD sistemine entegrasyonu
 - MSVD bileşenlerinin RMS ile çalıştırılabilir parçalara dönüştürülmesi
 - Dönüştürülen iş yüklerinin çalıştırmak için RMS'e gönderme altyapısının kurulması (Interaktif/Batch)
 - İş yüklerinin çalıştırılması
- PBM entegrasyonu
 - RMS yazılımının, Police PBM platformu ile entegre edilmesi
 - MSVD İş Yönetim Sisteminin PBM uyumlu hale getirilmesi
 - Yönetilecek parametrelerin belirlenmesi ve Politika tanımlanması
 - Interpolasyon yöntemi / karmaşıklığının ayarlanabilmesi
 - CPU başına MIPS, CPU doluluk oranı gibi parametrelerin belli değerlerine göre hesaplama sayısının adaptif ayarlanması
 - Kullanılacak kaynak miktarının (CPU, bellek miktarı) ve tipinin ayarlanabilir olması
 - Disk depolama mekanizmasının parametrelerinin ayarlanabilir yapılabilmesi
 - Belli kullanıcı tiplerinin sadece belli çözünürlük ve belli formattaki verilere erişmesine izin verilebilmesi
 - Görevlerin önceliklendirilebilmesi
 - Sistemle etkileşen kullanıcı sayısının kontrol altında tutulabilmesi
 - Erişim denetimi (Örneğin X düğümünün performans değeri $< X$ ise B tipinden kullanıcıların sisteme girişine izin verme)

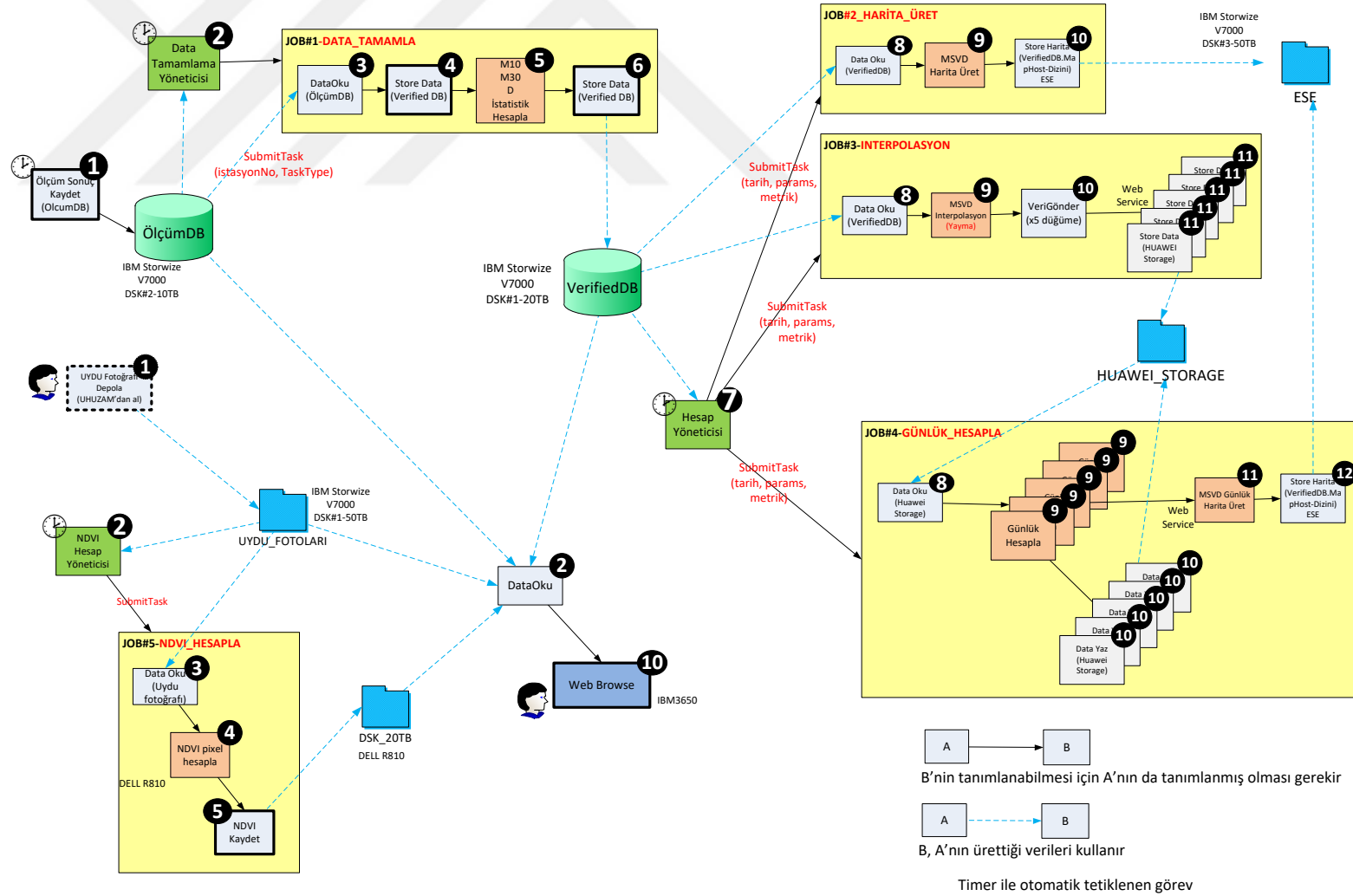
Şekil 6.15'deki hesaplama adımlarından birbiri ile ilişkili olanlar, bu şekilde ortak Task'ler içine gruplanmıştır. Bu incelemeler sonucunda Şekil 6.16 ve Şekil 6.17'de gösterildiği gibi MSVD bileşenlerinin pararele yük haline getirilebilmesi mümkün olmuştur.



Şekil 6.14 : TARBİL MSVD bileşenleri ve arayüzleri.



Şekil 6.15 : TARBİL MSVD süreç mimarisi ve öndelik ilişkileri.



Şekil 6.17 : MSVD için tasarlanan YBH mimarisi (sadeleştirilmiş iş listesi).

Tasarlanan mimarinin hayata geçirilmesi için yazılım gerçekleştirme çalışmaları sürdürülmüş ama tamamlanamamıştır.

6.3.1.2 TARBİL sistemi için YBH aracı seçimi

TARBİL altyapısında kullanmak üzere bir YBH platformu seçilmesi için çalışmalar sürdürülmüştür. Bu amaçla, güncel YBH yazılım platformu ürünleri, lisanslama, TARBİL bilgisayarlarına uyum, açık kaynak kod durumu, ölçeklenebilirlik, geliştirme durumu kriterlerine göre incelenmiştir. Açık literatürde bulunan aşağıdaki iki tabloya göre belirlenen HTCondor, PBS/Torque, Proactive ürünleri daha detaylı incelenmiştir.

Tasarım çalışmaları tamamlanmıştır. Ancak, gerçekleştirme çalışmalarının TARBİL MSVD yazılımında değişiklik gerektiren kısımları tamamlanamamıştır.

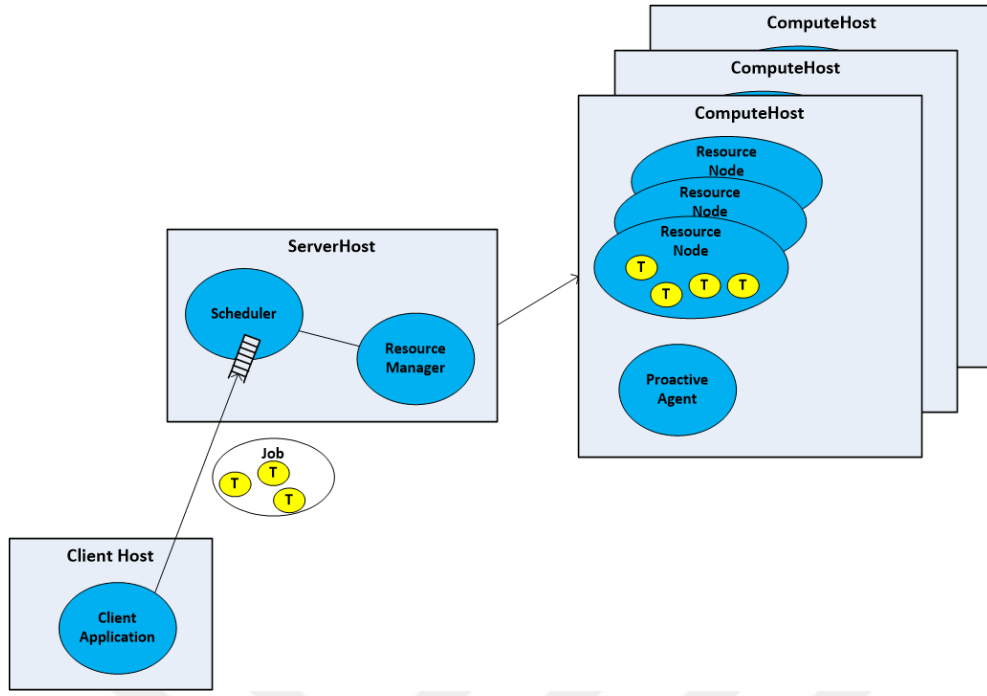
6.3.1.3 JMS araçlarının güncel durum araştırması

Bölüm 2'deki literatür taraması ile bu alandaki ürünler tanıtılmıştır. Bu bölümde, HPC/HTC alanında en çok kullanılan platformları belirleyip, aralarından TARBİL sisteminde kullanmaya en uygun olanı seçmek amaçlanmıştır. Tezin başında, zaten YBH sistemleri ile ilgili survey sonuçları ve açık literatürdeki (Wikipedia) JMS araçları karşılaştırma tabloları incelenerek ProActive JMS Platformu en uygun araç olarak seçilmiştir. ProActive, açık kaynak kodlu, ücretiz, geliştirmesi devam eden, HT ve HPC destekleyen, bütün işletim sistemleri ile uyumlu ve Police gibi Java'da geliştirilmiş olması belirleyici kriterler olmuştur.

6.3.1.4 Seçilen JMS aracı ve mimari tasarımı

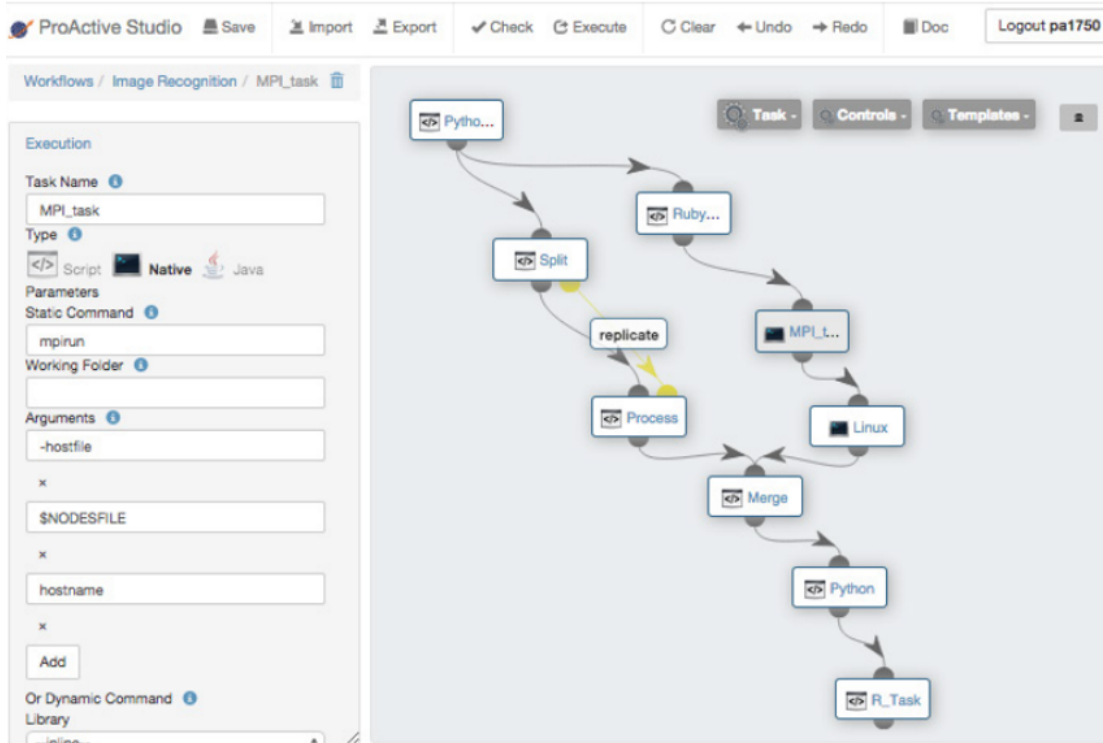
Proactive, ActiveEon, Inria ve OW2 konsorsiyumu tarafından aktif geliştirilmekte olan açık kaynak kodlu JMS platformudur [208]. Linux, Mac ve Windows işletim sistemleri için iş yükü yönetimi desteği sunar. Java dilinde geliştirilmiştir ve RESTful API'leri ile dış uygulamaların entegrasyonu kolayca sağlanabilir. Bu nedenlerle kendi geliştirdiğimiz PBM aracı ile entegrasyonu için tercih edilmiştir. Bütün işletim sistemlerini destekler. AGPL lisans ile ücretsiz kullanım mümkündür.

Bu platformda, iş yükleri (bir veya daha çok görev içeren) ve aralarındaki ilişkileri (DAG olarak) modellemek mümkündür. ProActive mimarisi, Active_object-based Java framework üzerine inşa edilmiştir. Bu sayede, görev dağıtımı optimize edilebilir ve hata dayanıklılığı sağlanır. ProActive mimarisi Şekil 6.18'de verilmiştir.



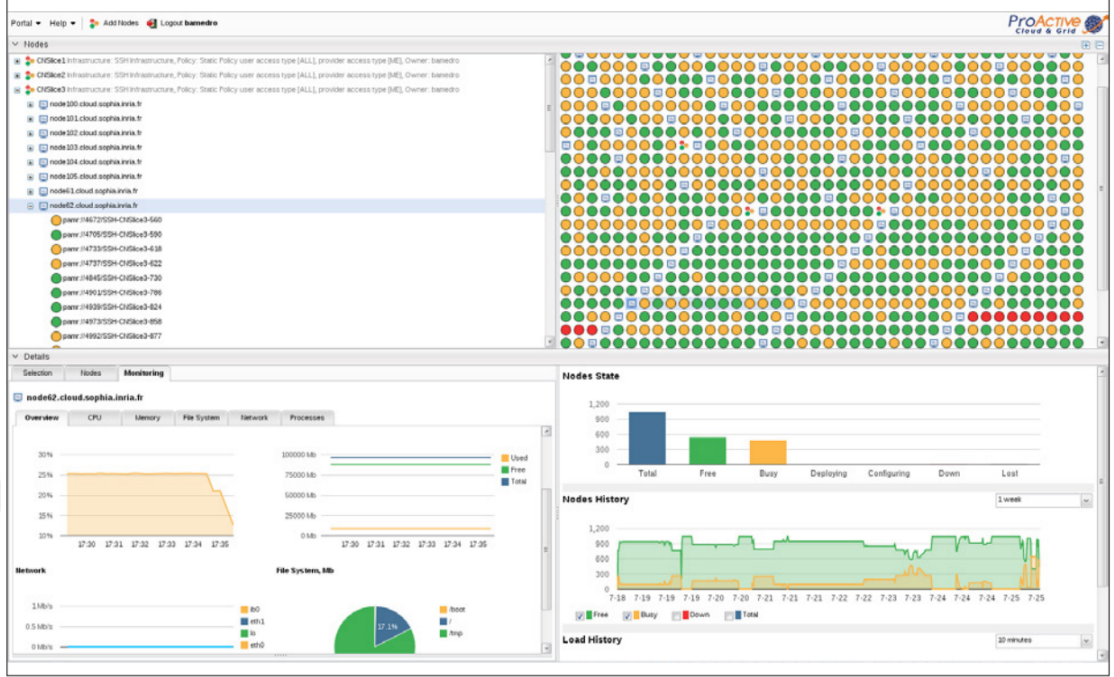
Şekil 6.18 : ProActive İş yükü yönetim bileşenleri.

İş yükü olarak, Java uygulamaları, script ve derlenmiş çalıştırılabilir uygulamaları kabul eder. İş yükleri, GPU desteği, kütüphane ve veri yoreselliği gibi çeşitli kriterlere göre kaynaklara atanabilir (Şekil 6.19 ve Şekil 6.20). Çeşitli tipteki (bulut, küme, PC) hesaplama kaynaklarını tek bir sanal altyapı gibi gösterir.



Şekil 6.19 : Workflow Studio: İş yüklerini bağımlılıkları tasarımı.

İş yüklerinin ve bağımlılıklarının görsel olarak tasarlanabilmesi çok sık iş yükü tasarımı yapılan sistemlerde kolaylık sağlar.

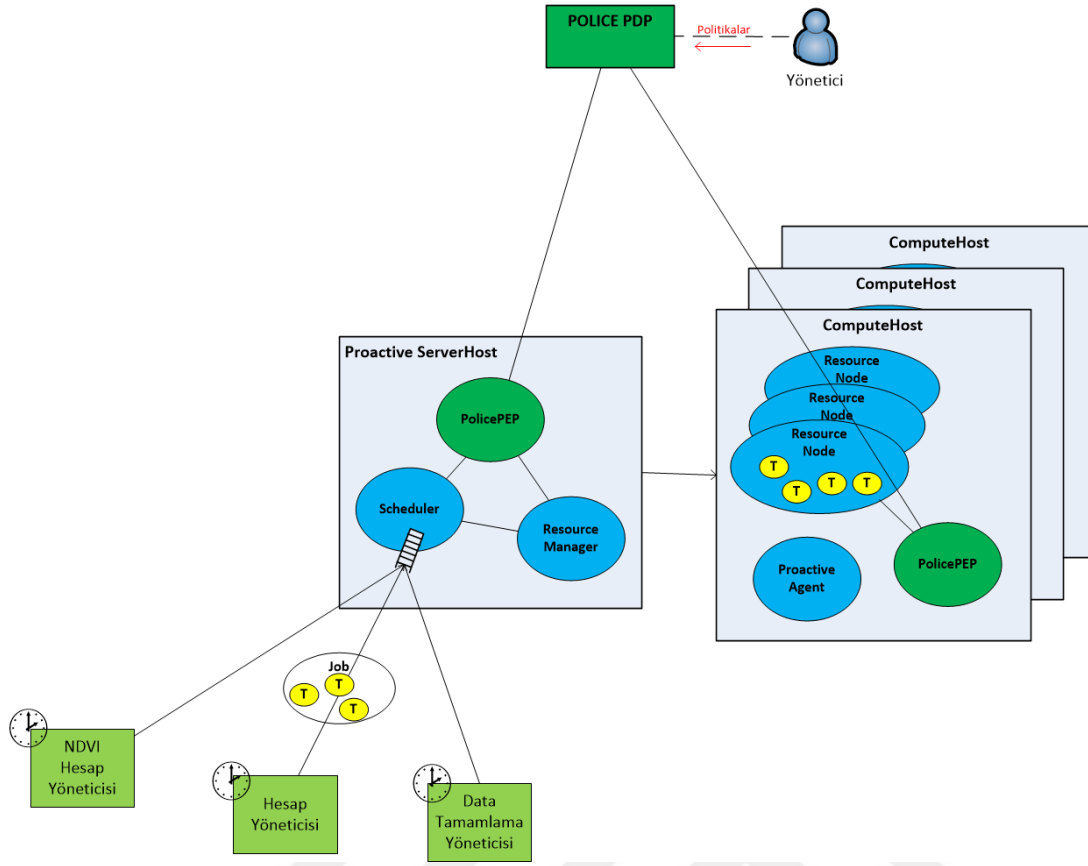


Şekil 6.20 : Resource Manager Portal: İş yükü izleme ve yönetme.

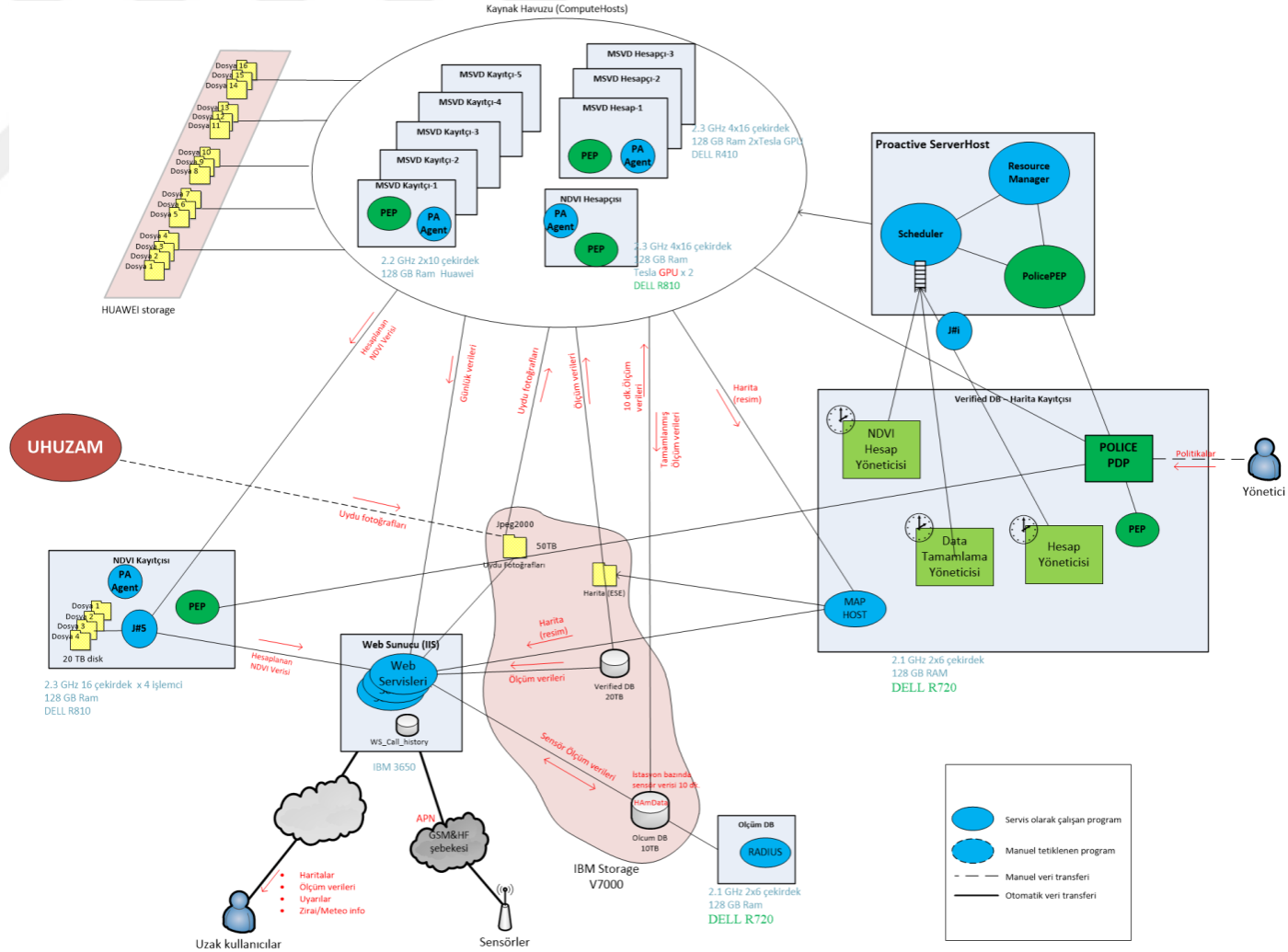
ProActive platformu, çalışan task'ler arasında çeşitli yollarla (bellek üzerinden, dosyalarla) veri geçişine izin verir. TARBİL MSVD yazılım bileşenleri arasında dosya paylaşımı, sıkça başvurulanan bir yöntemdir. ProActive, bu dosya paylaşımı ihtiyacının karşılanmasına yardımcı olur.

ProActive platformunun, arkasındaki akademik aktif destek, açık kaynak kodu, Windows ve Linux platformlarına olan desteği nedeniyle TARBİL MSVD iş yüklerinin HPC tabanlı olarak çalıştırma amacıyla kullanılmasına karar verilmiştir. Seçilen ProActive mimarisi ile Police PBM aracının entegrasyonu, Şekil 6.21'de gösterildiği gibi tasarlanmıştır.

Bu entegrasyon senaryosu, bütün MSVD sistemini yönetmek üzere devreye alındığında ise Şekil 6.22'deki politikalarla yönetilebilir sistem mimarisine ulaşılır.



Şekil 6.21 : ProActiveWS-POLICE PBM Entegrasyonu.



Şekil 6.22 : TARBİL MSVD'nin politika tabanlı yönetilebilir mimarisi.

Yapılan bu tasarımın hayata geçirilmesi için tez kapsamında aşağıdaki faaliyetler sürdürülmüştür.

- JMS platformu (Proactive) kurulumu ve konfigürasyonu
- ProActive ile POLICE PEP entegrasyonunun sağlanması

Tez süresi içinde tamamlanamayan aşağıdaki faaliyetler nedeniyle deneysel sonuçlar elde edilememiştir:

- Mevcut MSVD uygulamasının yeniden yazılarak İş Parçacıklarına dönüştürülmesi
 - Mevcut MSVD uygulamalarının yazılımları değiştirilerek TASK olarak paketlenmeleri
 - DataTamamlamaYöneticisi (Job Submitter) kodlanması
 - HesaplamaYöneticisi (Job Submitter) kodlanması
- Mevcut yük tiplerinin karakteristiğinin modellenmesi
- TARBİL yönetim politikalarının belirlenmesi ve sisteme uygulanması
- Politikalarla yönetilirken sistemin performans ölçümleri
- Politika tabanlı yönetilen sistem ölçümlerinin önceki durumla karşılaştırılması ve yorumlanması.

7. KAVRAM İSPATI ÇALIŞMASI-2: MERKEZİYETSİZ SİSTEMLERİN POLİTİKA TABANLI YÖNETİŞİMİ

7.1 Giriş

Blokszincirler, Merkeziyetsiz YBH sistemi sınıfının bir üyesidir. Diğer YBH sistemleri gibi, blokszincirler de farklı perspektiflerden (hata, konfigürasyon, varlık, performans ve güvenlik yönetimi) insan müdahalesine ihtiyaç duyar. Hızlı gelişen bir teknoloji olduğundan, blokszincirlerin henüz keşfedilmemiş sorunları ve iyileştirilmesi gereken konuları vardır. Mevcut blokszincir sistemleri genellikle sorunsuz çalışıyor olsa da zaman zaman yeni özellikler eklenmesi ve/veya acil hata düzeltmeleri gerekebilir. Öte yandan blokszincirler, farklı otoritelerin kontrolü altındaki düğümlerden oluşan merkezi olmayan bir mimariye sahiptir oldukları için yönetim süreçleri de geleneksel sistemlerinkinden farklıdır. Bu sebeple PBM kullanılabilirliğini göstermek için diğer YBH sistemlerinden farklı olarak iki farklı perspektiften problem ile yüzleşmek gerekir. Blokszincirlerinin yönetim gereksinimleri iki katmanda ele alınabilir: blokszincirin kurumsal varlığını ve egemenliğini sürdürmesini sağlamak (*blokszincir yönetiřimi*) ve blokszincir düğümlerinin kendilerinin yönetiřimi (*düğüm yönetimi*).

Blokszincir yönetiřimi teriminin yaygın bir tanımı, “*ortak katkıda bulundukları bir blokszincir projesi bağlamında paydařların yönünü, kontrolünü ve koordinasyonunu sağlama yöntemleri*” olarak verilmektedir [209]. Ancak, literatürde bu terimi farklı alt katmanlar altında ele alan başka yaklaşımlar da bulunmaktadır. Örneğin [210], blokszincir projelerinde yönetiřimi dört katmana ayırırken (altyapı, uygulama, řirket ve ülke), [209] yönetim katmanlarını zincir dışı topluluk, zincir dışı geliştirme ve zincir içi protokol olarak tanımlar. Blokszincir yönetiřimi, blokszincir'i ayakta tutmaya odaklanır. Bu kavram dört ana konuyu kapsamaktadır [210]: mutabakat, teřvikler, bilgi akışı, yönetim yapısı. Protokol kuralları, karar alma süreci, üyelik, roller, kayıt defteri geçmiři, dijital varlıklar gibi alt konulardaki kararlarla ilgilidir.

Daha dar kapsamlı olan düğüm yönetimi kavramı ise yalnızca blokzincir düğümlerin sahiplerini ilgilendirir. Yerel kaynakları ayrı ve bağımsız bilgi sistemleriymiş gibi yönetmek ile ilgilidir ve bu seviyedeki tüm kararlar düğüm sahibi tarafından verilir.

Aslında, durum kanalları (state channels), yan zincirler (side chains) ve blokzincirler arası iletişim gibi Layer2 konuları kapsayacak şekilde üçüncü bir yönetim katmanı oluşturulması da mümkündür. Zincirler arası birlikte çalışabilirliği yönetmeye odaklanmış [211] ilk çalışmalar ortaya çıkmaya başlamış olsa da bu katmandaki çalışmalar henüz başlangıç aşamasındadır.

Öte yandan blokzincir yönetimi, sadece blokzincirlerin kendisi ile ilgili bir konu değildir. [212]'de incelendiği gibi, blokzincir tabanlı bilgi sistem uygulamaları, genel giderleri (overhead costs) ve araçları azaltmak, birlikte çalışabilirliği ve verimliliği artırmak için hayata geçirilmektedir. Bu nedenle, bu uygulamaların üzerinde koştuğu blokzincir platformunun yönetiminin, bu koşan uygulamalar üzerinde, kullanıcılarını sansürlemek veya daha yüksek kullanım maliyetlerini zorlamak gibi çeşitli etkilere neden olabilir. Bu bağımlılık nedeniyle, yalnızca bilgi sistemlerinin sahipleri değil, aynı zamanda bazı senaryolarda kullanıcıları da yönetişime katılmak zorundadır.

Bir blokzincirin yönetimi, blokzincirin merkezi olmayan yapısı nedeniyle karmaşık bir konudur. Merkezi otoritelerin kontrolü altındaki diğer PDCA'lardan farklı olarak blokzincir, sistemin bir bütün olarak uzun vadeli sürdürülebilirliğini sağlamak için yeni dağıtık yönetim modellerine ihtiyaç duymaktadır. Blokzincir yönetimi ile ilgili yaptığımız literatür taramasından çıkardığımız [209-222], yönetimle ilgili üstesinden gelinmesi gereken zorluklar ve zorluklardan oluşan liste aşağıda verilmiştir. Bu liste, bütün seçenekleri içeren liste olmak amacıyla değil bilinen önemli zorlukları tanıtmak amacıyla hazırlanmıştır:

- Paydaşların mutabakatı. Çok sayıda paydaşı olan yönetim modelinde, tüm paydaşların çıkarlarını korumak için, uzlaştırmacı bir yol bulunması gerekmektedir. Bu ihtiyaç genellikle, çok aşamalı oylama süreçleri içeren, doğrudan demokrasi yaklaşımı olarak gerçekleşir. Bununla birlikte, oylamaya dayalı bu mutabakat yöntemi adil görünse de düşük seçmen katılımı, seçmenlerin oylanan seçenekler konusunda yeterli teknik bilgisinin olmaması, oyların rüşvetle alınması riski, kar maksimizasyonu amaçlayanların saldırıları

ve dominant uzmanların kararlarını taklit eden teknik bilgisi sınırlı katılımcılar vb. nedenlerle etkinliği konusunda hala endişeler vardır.

- Yönetişim güvenliği. Blokzincir sistemlerinin geniş saldırı yüzeyleri nedeniyle yönetim sürecinin güvence altına alınması da zor bir konudur. Aktörlerin yönetim faaliyetleri şeffaf olmalı ve aktörler hesap verebilir olmalıdır. Dürüst olmayan aktörler cezalandırılmalı veya yönetim sürecinden men edilmelidir. Ancak blokzincir'in, çok değişken, takma (pseudonymous) adlı, bazen de tamamen anonim kullanıcı kümesi vardır ve aktörlerin rolleri net olmayabilir. Sistem, yönetim sürecinde hile yapmak, kontrolü ele geçirmek, manipüle etmek, engellemek veya kötüye kullanmak için zayıflıklardan yararlanmak isteyen saldırganlara karşı dayanıklı olmalıdır. Örneğin, suistimal raporlama, kara liste ve yasaklı aktörlerin oylarının sayılmaması gibi kontrol mekanizmaları bulunmalıdır. Bu önlemler olsa da ne yazık ki oyları rüşvet ile yönlendirmek, doğru kararlar için hala zorlu bir tehdittir.
- Merkezileşme riski. Eksik teknik bilgiye sahip ortalama kullanıcılar, platform için iyileştirmeleri önermek ve savunmak yerine, karar alma sürecine katılmaktan kaçınabilir veya çoğunluğun veya uzmanların kararlarını taklit edebilir. Bu kaçınma halleri nedeniyle ortaya çıkan düşük katılım, blokzincir'in birkaç topluluk üyesi tarafından kolayca domine veya manipüle edilmesine neden olabilir. Büyük miktarlarda kripto varlığı olan paydaşlar veya madencilik tekelleri, diğer potansiyel merkezileşme nedenleridir. Blokzincir projesinin kurucuları, önerileri ön elemeyen editörler, ana geliştiriciler ve kendi çıkarlarını ön planda tutan madenciler gibi aşırı yetkilendirilmiş aktörler de blokzincirin merkezileşmesine yol açabilir.
- Adalet. Öncelikle kararlar objektif olmalı, blokzincir projesinin nihai hedeflerini ihlal etmemelidir ve sonuçları ile en fazla sayıda aktörü memnun etmelidir. Yönetişim kapsamında kayıt defteri (ledger) geçmişini değiştirmek, blokzincirin değişmezlik özelliğini (immutability) tehlikeye atabilir ve yönetime katılan seçmenlerin çoğunluğunun gücüyle ilgili endişeler yaratabilir. Kriptopara blokzincirlerinde, kaybolan kripto varlıklar için hak iddia edilecek güvenilir üçüncü taraflar veya merkezi yargı kurumları olmadığından, hiç kimsenin dijital varlığı haksız yere elinden alınmamalıdır. Ayrıca, güçlü kriptografik mahremiyet mekanizmaları içeren bazı

blokszincirlerde, haksız/adil olmayan işlemlerin geriye çevrilmesi, yönetim modelinde izin verilmiş olsa da mümkün bile olmayabilir. Karar alma süreci, mümkün olduğu kadar çok aktöre açık olmalıdır. Ayrıca, aktörlerin kararlar üzerindeki etkileri, dijital varlıklarının miktarları ve aktörlerin sistemdeki rolleri ile orantılı olmalıdır. Oylama gücünde merkezileşme ve bunun sonucunda ortaya çıkan eşitsizlik, adil yönetimin önündeki engellerdir. Bir diğer zorluk sürü psikolojisi (mob democracy) [217]. Kararlar, mantıktan ziyade duyular ve dedikodulara dayalı verilir. Oylamalar, *one person one vote* prensibine de uygun çalışmalıdır [214, 217].

- Teşviklerin hizalanması (Çelişen çıkarlar). Yönetim kararları, yönetim sürecine katılmak için farklı teşvik motivasyonlarına sahip, gevşek bağlı geniş topluluk üyeleri tarafından (global) olarak alınır. Ekosistemde, paydaşların çatışan çıkarları vardır. Örneğin, madencilerin yüksek ücret isteği, geliştiricilerin, ağ başarısını artırmanın yanı sıra değişikliklerin uygulanması üzerinde söz sahibi olma isteği, kullanıcıların, düşük kullanım maliyeti ve yüksek güvenlik talebi ve yatırımcıların ise varlık değerlerinin sürekli artması yönünde beklentisi vardır. Aktörler, genelde kendilerini ilgilendiren konulara daha fazla ilgi gösterirler. Ancak, düzgün işleyen bir yönetim için, topluluk üyelerine karar alma sürecine katılmaya yeterli teşvik sağlamak önemlidir. Motivasyon veya teşvik eksikliği nedeniyle, katılım hiçbir zaman topluluğun çoğunluğuna ulaşmazsa öneri oylamaları yeterince adil olmayabilir [216].
- Etkinlik. Yönetim prosedürleri, aşırı çaba, zaman ve kaynak gerektirmemelidir. Sürecin ve araçların karmaşıklığı etkinliği azaltır. Değişiklik önerisi yaşam döngüsü basit olmalıdır. Yönetim kararları bazen, sorunlu çatallaşmaların (contentious hard fork) blokszincir'i farklı kurallara sahip iki farklı blokszincir'e ayırmasına neden olur. Değişiklikler uygulandıktan sonra tek blokszincir olarak devam kararı alınsa bile çatallaşmalar genelde acı vericidir. Yönetim ve blokszincir'in kullanışlılığını azaltabilirler. Bütün düğümler için zorunlu yazılım güncellemesine sebep olurlar ve blokszincir genelinde kapsamlı bir çaba gerektirirler. Güncelleme kararlarının madencilere bırakıldığı bazı yönetim modellerinde, yeni yazılım sürümünün hiçbir zaman etkinleştirilmemesi ihtimali de vardır. Düğümlerin yeni yazılım sürümüyle güncellenmesi, blok üretiminde global veya yerel kesintilere neden olabilir.

Yönetişim sürecinin bizzat kendisi, sorunlu çatallaşma riskini artırmamalı ve kabul edilen değişikliklerin uygulanmasını kolaylaştırmalıdır.

- Karmaşıklık. Yönetişim prosedürleri basit olmalı, çok fazla adım ve farklı araç içermemelidir. Sıradan kullanıcılar için kullanımı kolay olmalıdır. Karmaşıklık ve kullanım zorlukları, hem katılımı azaltabilir hem de zaman ve kaynak ihtiyacını artırabilir. Ama mevcut modellerden özellikle off-chain yönetim modelleri, Github, Slack, Reddit, vb. pek çok harici sistem üzerinde yürüyen süreçler içerdiği için yürütülmesi karmaşık ve izlenebilirliği düşüktür.
- Yönetişimin esnekliği. Yönetişim sürecinin kendisi de statik değildir ve normal olarak zamanla gelişir. Bu nedenle, esnek olmalıdır. Ancak, yönetişimin mekanizmalarını ve kapsamını değiştirebilme yeteneği de zor ve riskli bir konudur. Yönetişim modelindeki radikal değişiklikler riskli olabilir, çünkü bir kez uygulandıktan sonra hatalı kararları geri almak daha zordur. Eğer değişiklik hatalıysa, daha hızlı ve kolayca istismar edilebilir.
- Kriz Durumunda Tepki süresi. Bir değişikliğe karar vermek ve bunu sahaya uygulamak için gereken toplam süre, özellikle acil durumlar için makul olmalıdır. Yavaş ve temkinli yönetim modelleri, (özellikle kripto para blokzincirlerinde tolere edilemezdir) kritik bir hatayı zamanında çözemeyebilir. Kritik hatalar, planlı bir oylamanın akışını bekleyemeyebilir.
- Değişiklik önerilerinin tutarlılığı. Değişiklik önerileri, blokzincir ağının ana hedefleriyle tutarlılık içinde olmalıdır. Ama bu o kadar kolay değildir. Genel olarak, yalnızca ana geliştiriciler, bu hedefler için endişelenir. Diğer aktör tiplerinden gelen öneriler ise çoğunlukla, bir problemi çözmeye veya bir iyileştirme önermeye yöneliktir. Bireysel teklif sahipleri yönetim kapsamında ele alınmış veya alınmakta olan her bir konuyu, ilişkileri ile izleyemeyebilir ve analiz edemeyebilir. Bu nedenle, muhtemelen birbiriyle çelişen veya geriye doğru uyumsuz birçok öneri sunabilirler. Yönetişim alanlarının birbirine girift yapısı nedeniyle öneriler arasındaki bağımlılıklar gözden kaçabilir ve bu bağımlılıklardan kaynaklanabilecek çelişkiler belirlenip çözüme kavuşturulmadıkça da sistemi istenmeyen durumlara sürükleyebilir.

Literatürde önerilen yönetim modelleri, yürütme biçimlerine göre zincir dışı (off-chain) ve zincir üstünde (on-chain) olmak üzere iki kategoride sınıflandırılmaktadır.

Zincir-dışı yönetim, doğrudan demokrasi gibi geleneksel sosyal yönetim yapılarına benzer. Ana geliştiricileri, madencileri, kullanıcıları ve iş dünyası aktörlerini, öncelikle değişiklik önerileri üzerinde uzlaştıran ve ardından blokzincir yazılımının manuel olarak güncellenmesini sağlayan bir prosedüre dahil eder. En iyi bilinen örnekler, Bitcoin BIP [223] ve Ethereum EIP [224] modelleridir. Zincir-dışı modeller, baskın aktörler nedeniyle nispeten merkezidir ve kararları etkilemek için yeterli teknik bilgi ve/veya finansal güce sahip olmayan birçok ana akım kullanıcıyı dışarıda bırakır. Blokzinciri dışında birçok faaliyet gerektirdiği için süreçler uzun sürer. Zincir dışındaki yönetim verileri nedeniyle oluşan denetlenebilirlik eksikliğinin yanı sıra radikal değişimler yerine küçük adımlarla tutucu bir şekilde, yavaş çalışır.

Zincir-üstünde (içinde) yönetim ise daha yeni bir kavram olup, temel farkı, yönetim modülünün, blokzincirin diğer bileşenleri gibi bütünleşik bir parçası olmasıdır. Böylece, *hard fork*'a ihtiyaç duymadan değişiklikleri yapmak mümkündür. Yönetim kurallarının ve kayıt defterinin değiştirilmesi mümkün olur. Mutabakat, genelde protokol üzerinde yürüyen oylamalar yoluyla sağlanır. Oylama sonuçları algoritmik olarak yönetilir ve sonuçların otomatik olarak uygulanması da doğrudan protokol tarafından gerçekleştirilir. Halihazırda zaten Tezos [225], Dfinity [226] ve Decred [227] gibi zincir-üstünde yönetim modelleri mevcuttur. Zincir-üstünde yönetim, zincir-dışı yönetimdeki bazı eksiklikleri karşılar.

Zincir-dışı yönetimin aksine, nispeten daha az merkezidir ve teknik bilgiye veya finansal güce sahip olmayan ana akım kullanıcıları da karar alma süreçlerine katar. Daha hızlı çalışır ve radikal değişiklikleri gerçekleştirme yeteneğine sahiptir. Süreçlerin ara adımlarına ait bilgilerin saklanması, onu daha şeffaf ve denetlenebilir hale getirir. Bununla birlikte, zincir-üstünde yönetim, kripto para alanında oldukça kutuplaştırıcı bir konu haline gelmiştir. Daha hızlı karar vermeyi sağlar, ancak risklidir. Çünkü yönetim sisteminin, kurulduktan sonra değiştirilmesi zordur. Hem potansiyelini [214] vurgulayan hem de endişeleri vurgulayan taraflar vardır [224, 216, 217].

Tezin bu bölümünde, mevcut modellerin karşılaştığı yukarıdaki zorluklardan bazılarına yanıt vermek için iki temel teknolojiyi birleştiren yeni bir zincir-üstünde yönetim modeli önerilmiştir. Bu amaçla öncelikle blokzincir platformlarının yönetimi ile ilgili zorlukları araştırılmıştır. Ardından, hem politika tabanlı Yönetim (PBM) hem de *merkezi olmayan kimlik* teknolojilerinin (DID), blokzincir yönetimi

sorunlarını adreslemek üzere uyarlanabilirliği, mevcut platformların halihazırdaki yönetim verilerine dayalı olarak analiz edilmiştir.

Ana katkı olarak, DID'ye dayalı yeni bir karar alma mekanizması ve önerileri politika formatında ele alma gibi bir dizi yeniliği içeren, yeni bir zincir-üstünde yönetim çerçevesi önerilmiştir. Kavram ispatı için, önerilen model, Ethereum blokzincir platformunda gerçekleşmiş ve deneyimler paylaşılmıştır.

Modelimizin içerdiği, paydaşların kolayca anlayabileceği üst düzey ifadelerle öneri tanımlamaya olanak sağlayan PBM teknolojisi, farklı bilgi seviyelerindeki aktörleri olan blokzincir ekosistemlerine iyi uyum sağlayacaktır. Politikaya dayalı tasarımıımız, mevcut modellerden yalnızca daha etkili değil, aynı zamanda daha basit ve daha kullanıcı dostudur. DID kilit teknolojisini de kullanan modelimiz, blokzincir yönetişiminin oy verme süreçleri ile ilgili endişelere de radikal bir çözüm önermektedir. Modelimiz, daha esnek bir demokrasi ve daha doğru yönlendirilmiş bir karar alma süreci sağlayabilecektir.

7.2 Benzer Çalışmalar

Blokzincirlerin yönetişimi için çeşitli çalışmalar yapılmış ve araçlar önerilmiştir. BIP ve EIP süreçleri gibi iyi bilinen zincir-dışı yönetim modellerine ek olarak, zincir-içi yönetim protokollerini uygulayan birkaç platform da vardır. Örneğin Tezos kripto para birimi [225], yerleşik bir yönetim modeline ve akıllı sözleşmelerin resmi matematiksel doğrulamasına sahip, merkezi olmayan bir akıllı sözleşme platformudur. Tezos, hisse kanıtı (PoS) işleyişine katılan kullanıcıların, kayıt geçmişini düzenleyebilmek de dahil olmak üzere her şeye oy vermelerine olanak tanır. Mutabakat sürecine katılmak için bir güvenlik teminatı gereklidir. Kullanıcıların oylarını başkalarına devretmelerine izin veren temsili demokrasiyi kullanır. Yönetişim önerisinde bulunma hakkı sadece, *Baker* denilen blokzinciri doğrulama düğümü işleten aktörlere verilmiştir. Diğer aktörler, bu önerileri, kripto varlıkları ile destekleyebilirler. Bir yönetim turu yaklaşık üç ay sürer. Her turda yalnızca bir değişiklik teklifi etkinleştirilebilir. Ön seçimle seçilen bir değişiklik önerisi için blokzincir üzerinde yürüyen bir dizi oylama yapılır. Yeterli katılım ve yüksek çoğunlukla kabul sağlanırsa (super-majority) önerilen güncelleme, 3 hafta boyunca bir test aşında işletilir. Ardından bir son onay oylaması (Promotion Vote) gerçekleştirildikten sonra, değişiklik otomatik olarak ana ağda devreye alınır. Öneriler,

değişiklikleri gerçekleştirmek için gerekli kaynak kodunu da içerir. Kod değişikliği önerileri yalnızca konsensüs protokolü bölümleriyle ilgili olabilir. Kabul edildiğinde, bu kod anında derlenir ve otomatik olarak etkinleştirilir. İlk başta, bir protokol değişikliğini öneren Baker, kendi düğümünde, bazı kontrolleri gerçekleştirip, değişikliği derleyip yükleyen yeni protokolü başarıyla devreye almalıdır.

Dfinity [226] kripto para platformu, *Blokszincir Sinir Sistemi* (BNS: Blockchain Nervous System) olarak adlandırılan algoritmik bir zincir-üzeri yönetim mekanizması kullanır. Dfinity, yönetim kurallarının değiştirilmesine ve kayıt defterinin kendisinde geriye dönük değişikliklere izin verir. Öneri türlerine göre değişen miktarda teminat mekanizması, spam teklifleri engellemek için kullanılmaktadır. BNS, insan tarafından kontrol edilen “nöron” adı verilen zincir dışı yazılımlar tarafından yapılan oylarla zincir dışı çalışır. Nöronlar, blokszincir üzerindeki BNS akıllı kontratını potansiyel değişiklik önerilerine karşı sürekli izlerler ve karar alma sürecine dahil olurlar. Bir aktör, kendisi BNS akıllı sözleşmesini çağırarak doğrudan oy verebileceği gibi likit demokrasi modeline (liquid democracy) benzer olarak başka bir aktörün oyunu taklit edebilir. Başka aktörlerin oylarını, öneri sınıflarına göre (Ekonomi, Politika, Protokol, İstemci Yükseltmeleri, Fixup/Freeze Resident). hazırlayacağı farklı takip listelerine göre taklit edebilir. Farklı değişiklik önerisi türleri için farklı oylama eşik değerleri kullanılabilir. Bir yönetim turu 28 gün sürer. BNS akıllı sözleşmesi, kabul edildiğinde değişiklik tekliflerini (akıllı sözleşmeleri dondurmak, token taşımak, kodu çalıştırmak, platformu yapılandırmak) yürütmek için EVM'deki ayrıcalıklı işlemleri kullanma yetkisine sahiptir.

Decred [227], diğerlerine göre daha karmaşık, özerk yönetim yapabilen bir başka (kısmi) zincir üstü yönetim modeline sahiptir. Hibrit bir konsensüs sistemi ile paydaşların kuralları koyduğu, merkezi olmayan ve kendi kendini yöneten bir para birimidir. Kullanıcılar, üç farklı yönetim etkinliğine katılmak için kripto varlıklarını kilitleyebilir. Bunlardan ikisi olan, PoW madencileri tarafından oluşturulan blokların PoS tabanlı oylama ile kullanıcılar tarafından onaylanması ve konsensüs kuralı değişikliklerinin oylanması, blokszincir üzerinde gerçekleştirilir. Bir konsensüs kuralı değişikliği için, yeni bir blokta, yükseltme kodunu da içeren bir DCP (Decred Change Proposal) önerisi gönderilir. Ardından, PoW madencileri ve PoS seçmenleri, oylamanın başlayabilmesi için yazılımlarını yeni sürüme yükseltmelidir. Yeni kodla yaklaşık bir haftalık çalışma sonrasında, yaklaşık bir ay boyunca PoS aktörlerinin

katılımıyla oylama gerçekleştirilir. Yeterli katılım sağlanamazsa veya çekimler dışındaki oylar süper çoğunluğa ulaşamazsa, oylama bir sonraki ay için de etkin kalır. Bir oylama, süper çoğunluğa ulaşmadan süresi dolarsa kapanır. Oylama kabul ile sonuçlanırsa, bir ay sonra (lock-in period) yeni konsensüs kodu aktifleştirilir. Lock-in adı verilen kilitleme periyodu boyunca, tüm düğümler yazılımlarını en son sürüme yükseltmelidir. Decred ekosisteminde kullanılan diğer yönetim mekanizması ise Decred Hazine fonlarının nasıl harcanacağı veya Decred anayasasına eklemeler gibi daha üst seviye konular için kullanılan Politeia portalidir. Politeia, Decred yönetim tekliflerini işlemek için sansüre dayanıklı, blokzincire linklenmiş/entegre şekilde işleyen, zincir-dışı bir yönetim sistemidir. Politeia oylamasıyla ilgili tüm veriler (ör. teklifler, yorumlar, oylar), periyodik olarak blokzincir'e sabitlenir ve Sybil saldırılarını ve kullanıcıların haksız sansürünü önlemek için kriptografik yöntemler kullanılır. Politeia oylaması bir hafta sürer. Sahte ve amaçsız öneri saldırılarına önlem olarak bir Politeia hesabı açmak ve değişiklik önerisi yapmak ücrete tabidir.

Zincir-dışı EIP ve BIP modelleri daha yavaş ve korumacıdır. İyileştirme önerilerini yavaş gerçekleyen ve uygulayan modellerdir. Zincir-içi modeller ise süreci hızlandırıyor gibi görünse de tamamen kusursuz değildir. Bir derece merkezileşme içerirler ve oylama yöntemleri yukarıda bahsedilen gereksinimlerin henüz tamamını karşılayamamaktadır. Örneğin hiçbirisi çok faktörlü mutabakatı desteklemez ve hepsinin, takip etmesi kolay olmayan birçok aşamadan oluşan, uzun ve karmaşık süreçleri vardır. Dahası, EIP ve BIP editörlerinin olduğu gibi, Politeia yöneticisinin de bir önerinin oylanmasını engelleme yetkisi bulunmaktadır.

Bu yönetim modellerinde, aktörlerin gerçek kimlikleri yalnızca forumlar, portallar vb. gibi zincir dışı bir ortamda gerçekleştirilen tartışmalar sırasında bilinir. Kalan aşamalarda ise genellikle takma adlar (pseudonym) kullanılır. Aktörlerin oy verme yetkinliklerini doğrulayamama ve hesap verebilirlik eksiklikleri yüzünden, tüm mevcut yönetim modelleri, yalnızca finansmana dayalı güvenlik sağlayan hisse bazlı oylama planları ile donatılmıştır.

Çalışmamız, bildiğimiz kadarıyla, PBM ve merkezi olmayan kimlik teknolojilerini blokzincir yönetimine dahil etmeyi amaçlayan ilk araştırmadır. Çizelge 7.1, mevcut yönetim modellerinin ve çalışmamızın yönetim ile ilgili zorluk ve güçlükleri nasıl ele aldığını karşılaştırmaktadır.

Çizelge 7.1 : Yönetişim zorluklarına göre modellerin karşılaştırılması.

Zorluk ve güçlükler	Zincir-dışı modeller (BIP, EIP, ve diğerleri)	Zincir-içi modeller (Tezos, Decred, DFINITY)	Tezde önerilen model
Paydaşların konsensüsü	- Teklif için editör ve ana geliştiricilerin onayı - Sınırlı sayıda aktörle zincir dışı oylama - Madenciler yükseltmeyi seçip seçmeme yetkisine sahip	- Herkese açık, ancak aktör türlerini içermeyen çok aşamalı zincir-içi oylama - Likit demokrasi	- Herkese açık çok faktörlü bir dizi oylama - DID destekli likit demokrasi
Yönetişim güvenliği	- Sahte aktör <i>bir-oy-bir-kşi</i> ilkesini ihlal edebilir - Editörler tarafından teklif ön seçimi - Madenciler denge mekanizmasıdır - Rüşvet tehditi vardır - Konsensüs detayı herkese görünür olmayabilir - Güncelleme toplantılarının tutanakları aracılığıyla sınırlı şeffaflık - Kimse sorumlu tutulamıyor	- Sahte aktör <i>bir-oy-bir-kşi</i> ilkesini ihlal edebilir - Para kilitleyerek oylama. Dürüst olmayan aktörlerin payını almak - Rüşvet saldırılarından daha ağır basacak kadar hisse sahibi - Aşamaların tüm ayrıntıları (teklifler, oylama) zincir üzerinde herkese açıktır.	- Güven zinciri ile DID'leri kaydedilmiş aktörler - DID açık anahtarları ve VC bilgileri aracılığıyla aktörlerin kimlik doğrulaması ve yetkilendirilmesi - Sybil saldırılarını önlemek için <i>nitelikli bakiyeler</i> - DID'lere dayalı zincir-içi oylar - Kötüye kullanım raporlaması ve yönetimden kimlik tabanlı yasaklama
Merkezileşme riski	- Editörlerin (filtreleme teklifleri), ana geliştiricilerin (karar verme için) ve madencilerin (sahaya uygulama için) orantısız gücü. - Ethereum ve EOS gibi vakıfların hakimiyeti.	- Tüm paydaşlara açık - Zincirleme oylamalar - Uzmanlar veya balinalar nedeniyle düşük katılım ve tekelleşme riskleri - Kullanıcılar madencilere ve ana geliştiricilere karşı güçlüdür	- Tüm paydaşlara açık ve sadeliği sayesinde cezbeder - Çok faktörlü konsensüs merkezileşmeyi önler
Adillik	- Editörler aşırı güçlüdür ve teklifleri geciktirebilir veya reddedebilir - Sınırlı sayıda aktör karar verir - Oy gücündeki eşitsizlik	- Defter'i oylamayla değiştirmenin sürdürülebilirliği ve adillliği henüz kanıtlanmamıştır. - Oy gücündeki eşitsizlik - Takma isimle <i>bir-oy-bir-kşi</i> ihlal edilebilir	- Daha fazla oyuncu çekiyor - Oylar, rollere ve bakiyelere göre ağırlıklandırılır - Nitelikli bakiyeler - Rollerin çoğu için <i>bir-oy-bir-kşi</i> ihlal edilemez

Çizelge 7.1 (devam): Yönetişim zorluklarına göre modellerin karşılaştırılması.

Zorluk ve güçlükler	Zincir-dışı modeller (BIP, EIP, ve diğerleri)	Zincir-içi modeller (Tezos, Decred, DFINITY)	Tezde önerilen model
Teşviklerin hizalanması (Çakışan çıkarlar)	- Sınırlı sayıda aktör karar alır ve uygular - Madenciler kararları uygulamayı tercih etmeyebilir	Çıkarlar, bilgi eksikliği, hisse eksikliği nedeniyle düşük katılım riski	Politika temelli öneriler, kolay anlaşılır olması nedeniyle katılım motivasyonunu artırabilir
Etkinlik	- Farklı araçlar ve aktörlerle deterministik olmayan aşamalar - Yükseltme kararının madencilere bırakılması, hard fork olasılığını artırır	- Sadece küçük topluluklarda iyi çalıştığı söyleniyor. - Otomatik kod aktivasyonu - Değişiklikleri otomatik olarak etkinleştirmek, insanları şüpheli yapar.	- Tek tip ve nispeten hafif süreç - Politika olarak teklif etkinleştirmeleri - Azaltılmış hard fork riski
Karmaşıklık	- Farklı araçlar ve platformlar - Bir önerinin referans gerçekleştirmesini anlamak kolay değil	- Kullanım kolaylığı için aşamaların daha basitleştirilmesi gerekir. - Karmaşık öneriler kolay anlaşılır değildir.	- Politika önerilerinin anlaşılması kolaydır. - Tek araçla sağlanır - Hafızalı yönetim ile işlenen öneriler
Esnek yönetim	- Katı (Esnek değil) - Araçları ve süreci değiştirmek zor	- Zincir dışı modellerden daha esnek - Yönetişim konuları sabittir	- Yönetişimin kendisi de yönetişimin konusu - Politikalarla yönetim daha kontrol edilebilir ve daha esnektir
Acil durumlarda yanıt süresi	- Kademeli iyileştirmelerle yavaş ve tutucu çalışır - Belirsiz aktivasyon tarihi	- Oldukça yavaş ve çok aşamalı - Genel olarak belirli süreli, planlanmış oylama yapılır	- Yazılım yükseltmesi gerektirmeyen politika tabanlı öneriler sayesinde çevik karar verme - Birden fazla öneri paralel olarak işlenebilir
Teklifler arası bağımlılıklar	İnsan müdahalesi ile yönetilir	İnsan müdahalesi ile yönetilir	Çatışmalar, otomatikleştirilmiş PBM yazılımı tarafından ele alınır

7.3 Arka Plan

Bu bölüm, modelimizin üzerine inşa edildiği temel yapı taşlarını (yani politikaya dayalı yönetim ve merkezi olmayan kimlik kavramları) tanıtmaktadır. Gerçek dünya verilerini analiz ederek bu temel yapı taşlarının, yönetim alanına nasıl uygun olduğunu irdelemektedir.

7.3.1 PBM'in blokzincir yönetimi için kullanılabilirliği

Düğümün yönetimi, hem basit ve sıradan olduğundan hem de zaten on yıllardır endüstrinin üzerinde çalıştığı bir konu olduğundan, bu tez çalışmasını kapsamı dışında bırakılmıştır. Diğer yandan, blokzincir yönetimi konularında PBM teknolojisini kullanmak ise hem öncü bir çalışma olma hem de pek çok sorunu çözme potansiyeline sahiptir. Tezde, değişiklik önerilerinin merkezi bir otorite tarafından değil, yönetim modeli içerisinde oybirliği ile kararlaştırılan politikalar olarak uygulanmasını önerilmiştir. Aşağıdaki liste, PBM teknolojisinin blokzincir yönetiminin gereksinimlerini karşılayıp karşılamadığına dair bir analiz sunmaktadır:

- Yönetimin en önemli özelliği, topluluğu tarafından belirlenen üst düzey hedeflenen amaçlara uygun olarak bir blokzinciri yönlendirmektir. PBM'nin felsefesi tam da bu ihtiyacı karşılamaya uygundur.
- PBM, paydaşlar tarafından kolayca anlaşılabilir üst düzey ifadelerle bir sistemin davranışını kontrol etmeye izin verdiğinden, farklı bilgi düzeylerinden aktörlere sahip blokzincir ekosistemlerine iyi uyum sağlayacaktır.
- Blokzincirler herhangi bir nedenle oluşacak hizmet kesintilerine tolerans gösteremez. Bu nedenle, hard-fork'ların veya soft-fork'ların neden olduğu kesintiler veya genel giderlerden kaçınılmalıdır. PBM'nin bir blokzincir düğümünün davranışını ve hatta akıllı sözleşmeleri durdurmadan değiştirme yeteneği bu amaca hizmet edebilir.
- Politikalar hep birlikte, istenen davranış formal olarak tanımlayan ve bu davranışta yapılmak istenen değişikliklerin uyumluluğunun da yazılımla formal olarak doğrulanabildiği dijital anayasalar olarak görülebilir [220]. Yani politikalar, daha tutarlı yönetim için otomatik olarak doğrulanabilir bir dijital anayasa işlevi görebilir.

- Etkinleştirildikten sonra değiştirilmesi zor olduğundan yönetim prosedürünü değiştirmek risklidir. Eğer hatalıysa daha hızlı ve kolay bir şekilde istismar edilebilir veya oynanabilir. Ancak yönetimin politikalarla uygulanması, mevcut katı modellerden daha kontrol edilebilir, daha çevik ve daha esnektir.
- Ortalama aktörler için politikaları anlamak, koda dayalı değişiklik teklifleri ve bunlar üzerindeki tartışmalardan daha kolaydır. Böylece daha fazla insan başkalarının fikirlerine uymak yerine kendi özgün fikirleriyle kararlara katılabilir.
- Kritik bir hata, mutabakata dayalı bir yönetim prosedürü ile hızlı olarak çözülmeyebilir. Politikalar ise çevik karar almaya ve hızlı müdahalelere yardımcı olarak değişikliklerin hızlı etkinleştirilmesine izin verebilir.
- Mevcut yönetim modellerinin bazılarında, karar verme, implementasyon ve uygulama(deployment) aşamaları ayrı süreçlerdir ve madencilerin yeni yazılımı etkinleştirmek için mutabakata varamaması durumunda, devreye alınması gerçekleştirilemeyebilir. Ayrıca, oylama aşamasından sonra güncellemeler yapılırsa, bu uygulamaların kabul edilen teklifle tutarlılığının sağlanması da bir başka açık konudur. Politika tabanlı yönetim ise güncellemeleri, oylama döneminin bitiminden sonra ek bir işleme gerek kalmadan hemen uygular.
- Diğer bir avantaj, güncellemelerin geliştirilmesi sırasında test kolaylığıdır. Politikalar aracılığıyla çeşitli parametreleri ayarlamak, hem yeni kod oluşturmaktan hem de her değişiklik kümesi için bu parametreleri düğümlere yüklemekten daha kolaydır.
- PBM, olası anlamsal bağımlılıkları ve teklifler arasındaki çelişkileri ele almak için birçok araç ve şema kullanma potansiyeline de sahiptir.
- Ayrıca PBM, eski bilgi sistemlerine daha yakın bir sınıf olan özel veya izin verilen blokzincir ağlarının yönetimine daha uygundur.

Yönetim politikaları kapsamında ele alınabilecek parametreler ve eylemler, protokol (ör. blok boyutu, işlem boyutu, doğrulama kuralları, erişim kontrolü, zorluk seviyesi vb.), iş modeli (ör. blok ödülü, ücret, proje fonları vb.) ve yönetim prosedürünün kendisidir (örn. oylama eşikleri, izin verilen değişiklikler vb.). Önerimizi ve yukarıda sıralananları haklı çıkarmak için gerçek dünyadan örnekler incelenmiştir. Sınıflarında

en yüksek teklif sayısına sahip Bitcoin [223], Ethereum [224] ve Decred [227] platformlarının iyileştirme önerilerinin, Politika olarak uygulanıp uygulanamayacaklarını anlamak için tek tek analizi yapılmıştır. Çizelge 7.2'de verildiği üzere, önerilerin önemli bir kısmının PBM aracılığıyla ele alınabilirliği (“KapsamDışı” sütununu göz ardı edersek) görülmüştür.

Çizelge 7.2 : Gerçek değişiklik önerilerinin modelimizle gerçekleştirilebilirliği.

Sistem	Yönetişim Dönemi	Toplam Öneri Sayısı	PBM		
			Kullanılabilir	Kullanılamaz	Kapsam Dışı
BIP (Bitcoin Improvement Proposal)	10.01.2011-20.03.2020 (3.357 gün)	137	27	53	57
EIP (Ethereum Improvement Proposal)	15.11.2015-16.06.2020 (1.675 gün)	264	48	76	140
DIP (Decred Improvement Proposal)	04.05.2017-19.06.2020 (1.142 gün)	88	4	7	77

Bu analize ait ayrıntılı veriler [257] 'de paylaşılmıştır. PBM'nin yetersiz kaldığı çoğu durum, konsensüs protokolünü değiştirmek ve yeni kripto algoritmalarına geçmek gibi büyük değişikliklerdir. Bu durumlarda, PBM daha çok mevcut yönetim araçlarına yardımcı rolde kullanılabilir. Her şeye rağmen, soft fork olarak yapılmış önemli sayıda öneri, PBM aracılığıyla maliyet etkin ve hızlı bir şekilde uygulanabilir bulunmuştur.

7.3.2 DID yönetim teknolojisi ve modelimizde kullanılma gerekçesi

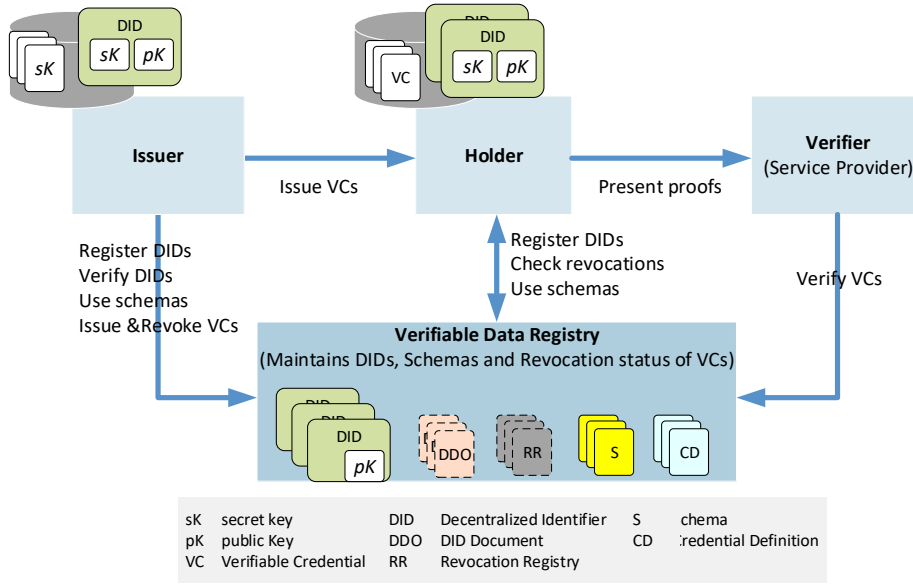
Bir Kimlik Yönetim (IdM) sisteminin [228] temel işlevi, her kişinin gerçekte iddia ettiği kişi olduğunu doğrulayabilmek için bir tekil tanımlayıcıyı ve bazı nitelikleri gerçek dünyadaki bir varlığa güvenli bir şekilde bağlamaktır. Mevcut sistemlerin çoğu merkezi veya federe IdM'lerdir ve tüm aktörler merkezi kimlik sağlayıcılarına güvenmek zorundadır. Son yıllarda ortaya çıkan, blokzincirin araçları ortadan kaldırma yeteneği [30] IdM alanında çok fazla dikkat çekmiştir. Bu durum, bireylerin kimlikleri üzerinde tam kontrole sahip oldukları, tamamen merkezi olmayan SSI (Self-Sovereign Identity) [229, 230] yönetim çerçevelerinin uygulanmasını kolaylaştırmıştır. Önerilen blokzincir tabanlı IdM sistemlerinin çoğu akademik dünyadandır [231-233]. Bazıları, ayrıca IdM'de mahremiyeti koruma kavramını

gündeme getirmiştir [234-239]. Akademinin yanı sıra, endüstri tarafında, blokzincir tabanlı IdM için ticari ürünler ve defakto standartlar oluşturma çabaları olmuştur. uPort [240], Shocard [241], Bitnation [242], Civic [243], Jolocom [244], Sovrin [245], Evernym [246], DIF ve ID2020 Alliance [247] gibi girişimler ve BT aktörleri, çoğunlukla ERC'ler [248] ve W3C Spesifikasyonları [249] gibi ortaya çıkan standartlara dayanan merkezi olmayan IdM'lerin geliştirilmesine odaklanmıştır.

Dijital kimlikler, güvenilir olmayan ortamlarda, varlıklar arasındaki etkileşimleri kolaylaştırır. W3C, genel kabul gören, merkezi olmayan tanımlayıcılar/kimlikler (DID'ler) için ortak bir veri modeli, formatı ve işlemleri [250] için spesifikasyonlar yayınlamıştır. Bunlar, Aktörlerin istedikleri kadar benzersiz DID'lere [250] sahip olmalarına izin verir. Her bir DID için bir asimetrik anahtar çifti (sK , pK) üretilir ve DDO (DID Document) adlı, DID'in nasıl kullanılacağını tanımlayan bir meta bilgi ilişkilendirilebilir. DDO, kimlik sahibinin açık anahtar(lar)ını (pKs) ve kimlik sahibiyle iletişime hangi sanal adresler kullanılarak geçilebileceği bilgilerini içerir. Kimlik sahibi, diğer aktörler ile etkileşimde kendi özel anahtarını (sK) kullanır. sK , blokzincir dışında, kimlik sahibinin kontrolündeki bir cüzdan yazılımında depolanır.

Bir aktörün özelliklerini belirten, kişisel olarak tanımlanabilir bilgiler, *Doğrulanabilir Kimlik Bilgileri* (VCs: Verifiable Credentials) spesifikasyonları yoluyla gelir [251]. VC, bir yayıncı (ör. güvenilir bir kuruluş) tarafından kriptografik olarak imzalanmış, bir aktör hakkında kurcalamaya dayanıklı iddialar grubudur. Kimlik bilgilerinin, kriptografik olarak güvenli, mahremiyet odaklı ve otomatik olarak doğrulanabilir, standart bir formatta ifade edilmesini sağlar. Bir blokzincir (Doğrulanabilir Veri Deposu), VC'leri kullanmak için gerekli olabilecek olan, VC şemaları, kimlik bilgi tanımları, iptal kayıtları, DID, DDO vb., verilerin oluşturulmasına, depolanmasına ve doğrulanmasına aracılık eder.

DPKI (decentralized PKI) [252] olarak adlandırılan, DID ve VC'lere dayalı olan, genel kabul görmüş bu model Şekil 7.1'de gösterilmiştir. Bu modelde, *Subject*, hakkında nitelik iddialar (claims) düzenlenen varlıktır. *Holder*, bir veya daha çok DID değerine ve ilişkili VC bilgilerine sahip olan ve bunları, başka aktörlere ispatlayabilen aktördür. *Issuer*, *Subject* hakkında nitelik sertifikası (VC) hazırlayan, içeriğini tasdik eden varlıktır. *Verifier* (Doğrulayıcı), sunduğu elektronik hizmetlerine erişmeye çalışan aktörlerden belli içerikteki VC'lerini talep eden ve doğrulayan aktördür.



Şekil 7.1 : Blokzincir tabanlı DID çözümlerinde roller ve bilgi akışları [230].

Doğrulayıcı, blokzincir'den, ilgili Issuer'ın *pK* ve Revocation Registry bilgilerine ulaşarak, kendisine sunulmuş VC'lerin, güvendiği bir otorite tarafından yayımlandığını, değiştirilmediğini ve halen geçerli olduğunu kriptografik olarak doğrulayabilir.

Sistemde Yayınlayıcı, Holder veya Doğrulayıcı olarak hareket etmek, herhangi bir otoritenin onayına tabi değildir. Herkes blokzincirini okuyabilir veya yazabilir ve neyi kiminle paylaşacağına kendisi karar verebilir.

Blokzincirde oylama süreçleriyle ilgili birçok endişe dile getirilmiştir. Artan endişelere rağmen, coin oylaması hala yönetim için Sybil'e dirençli bir mekanizma olarak görülmektedir. Oylama, ayrıca çekirdek geliştirme ekibine karşı bir denge mekanizması olabilir. Bununla birlikte, aşağıda alıntıladığımız iyileştirme önerileri de mevcuttur. Örneğin, kararın farklı mekanizmaların ve grupların toplam sonucuna bağlı olduğu *çok faktörlü mutabakat* (multi factor consensus) metodolojisi [216, 219] iyileştirme için kullanılabilir. Bu mekanizmalar aşağıdakileri içerebilir:

- Projenin yol haritası (örneğin, projenin gideceği yön hakkında yayınlanan fikirler listesi)
- Baskın çekirdek geliştirme ekipleri arasında mutabakat
- Kripto varlık sahiplerinin oyları
- Kullanıcıların oyları (Sybil-resistant yoklama mekanizmaları ile)
- Yerleşik normlar

Çok faktörlü mutabakat, likit demokrasi [253] ve *kuadratik oylama* [254] gibi oylama yöntemleri, doğrulanabilir kimliklere dayanır ve paydaşların kendilerini yanlış tanıtılmalarını önlemek için bir mekanizma gerektirir [255]. Aktörlerin daha fazla etkilendikleri konularda oy kullanmalarına izin vererek kontrollü oylama kullanmak faydalı olabilir [216]. Çok faktörlü mutabakat gibi öneriler, Sybil saldırılarına karşı direnç sağlayabilmek için aktörlerin rolleri hakkında kesin bilgi gerektirir. Ancak merkezi olmayan (merkeziyetsiz) dünyada, gerçek dünya kimliklerini blokzincire bağlamak, güvenilir üçüncü tarafların eksikliği nedeniyle kolay değildir. Yakın geçmişte ortaya çıkan DID kavramı [230], her seçmenin rolünün kesinlikle tanınmasını kolaylaştırırken mahremiyeti de koruyan bir çözümdür. Böylece, yönetişimde daha esnek bir demokrasi anlamına gelen çok faktörlü mutabakat yeteneğini mümkün kılabilir. Ayrıca doğrudan, temsili ve likit demokrasi gibi tüm demokrasi gerçekleştirme yöntemlerini de destekleyebilir.

7.4 Önerilen Model

Bu bölümde, modelimize mimarisel bir genel bakış sunulmakta ve çeşitli uygulama ayrıntıları tartışılmaktadır. Öncelikle, hangi teknolojiyi hangi amaçla kullandığımıza ilişkin bir genel bakış sağlanmakta, ardından da ana bileşenleri ve görevlerini vurgulayarak sistem mimarisi tanıtılmaktadır. Kabul edilen yönetim önerilerinin aktivasyon sürecinin açıklanmasının ardından, modelin DID bazlı oylama şeması detaylı olarak açıklanmaktadır. Uygulama ve sonuçlar bölümünden sonra, kurulan bir saldırı modeli üzerinde sistemimizin tehditlere nasıl yanıt verdiği irdelenmektedir.

7.4.1 Genel bakış

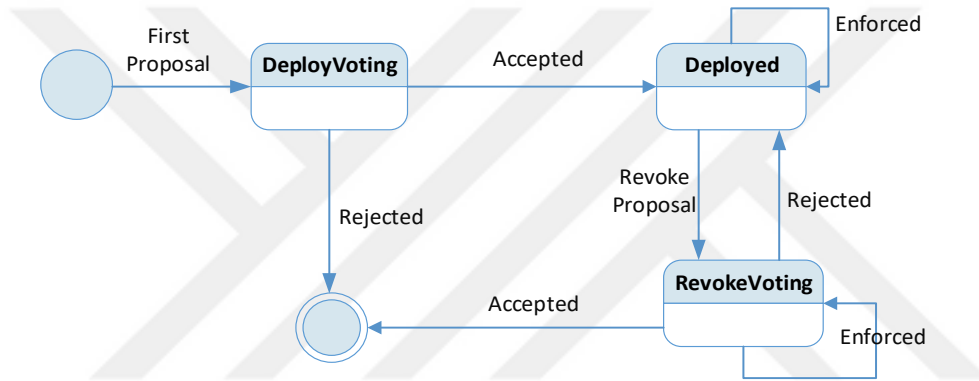
PBM ve DID [228, 250] kavramlarına dayanan bir zincir-içi blokzincir yönetim modeli sunulmuştur. Modelin tasarımında, daha basit, adil, sağlam, genişletilebilir, kullanımı kolay ve esnek bir çerçeve oluşturmaya odaklanılmıştır. Modelimiz, mevcut yönetim modelleriyle benzer yetenekler içerirken, bazı yönetim zorluklarını doğrudan çözmekte ve diğerlerinin ise çözümü için bir altyapı sağlamaktadır.

Yönetişim modelimizde, değişiklik önerileri politika biçimindedir ve blokzincir düğümleri, bu politikaları uygulamak için PBM yeteneği ile donatılmıştır. Politikalar, yönetim sürecini çok daha basit, kullanımı kolay ve esnek hale getirir. Aynı zamanda

yönetişime yüksek katılımı da sağlayabilir. Ayrıca, politikalar, daha tutarlı yönetim için otomatik olarak birlikte doğrulanabilir bir dijital anayasa işlevi görebilir.

Öneriler, yeni bir politika önermek veya mevcut bir politikayı iptal etmek amacıyla sunulur. İlgili değişiklik önerisinin yaşam döngüsüne karşı düşen bir politikanın yaşam döngüsü Şekil 7.2'de gösterilmiştir. Bu politika değişikliği, mutabakat sağlanırsa sisteme otomatik olarak (blokzinciri durdurmadan) uygulanır.

Bir blokzincir düğümünün herhangi bir bileşeni, yönetilen bir nesne rolünde, PEP ile entegre olursa politikalar ile yönetilebilir. Mutabakat ve ağ iletişimini içeren blokzincir protokolü, işlemler, hesaplar, akıllı sözleşmeler, durum verileri gibi dijital varlıklar, hatta yönetim sürecinin kendisi, politikalar aracılığıyla yönetilebilir.



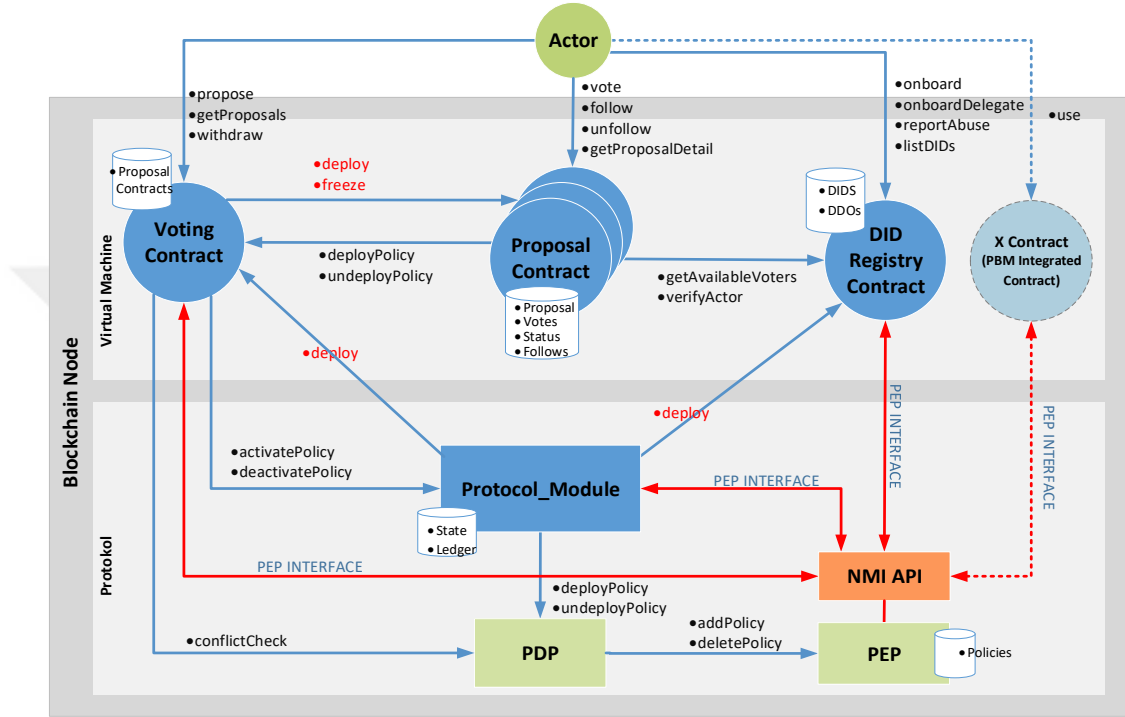
Şekil 7.2 : Önerilen modelin politika yaşam döngüsü.

Diğer yönetim modellerine benzer şekilde, oylama kavramı, yönetim aktörlerinin mutabakat aracı olarak kullanılır. Ancak oylama modüllerinin tasarımı sırasında Bölüm 7.1'de tartışılan eleştirileri ve Bölüm 7.3.2'de belirtilen iyileştirme önerileri dikkate alınmıştır. Çok faktörlü mutabakat kavramına dayanan daha esnek bir demokrasi modeli oluşturmak için, politika önerilerini kimlerin sunabileceğini ve sunulan önerilerin nasıl değerlendirileceğini yönetmek için DID teknolojisinden yararlanılmıştır. Aslında, rol bazlı oylamayı destekleyen yöntemimiz, sadece değişiklik önerilerinin değil, aynı zamanda söz konusu blokzincir ile ilgili herhangi bir konuda kararların alınmasında da kullanılabilir.

7.4.2 Mimari modeli

Şekil 7.3'te gösterildiği gibi, sistem mimarisi iki katmanda bileşenler içerir. Yönetişim kodu, bir sanal makinede bir dizi akıllı sözleşme olarak uygulanırken, diğer önemli bileşenler, blokzincir düğümünün protokol katmanına gömülüdür. Ayrıca aktörlerin,

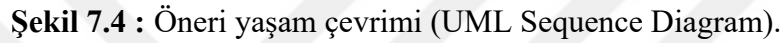
DID ve diğer hesap anahtarlarını depolayan zincir dışı bir cüzdan bileşeni de vardır. Basitlik adına, bir blokzincir düğümünün diğer tüm bileşenlerini *Protocol_Module* adlı bileşen ile temsil edilmiştir. Genesis bloğunda sisteme konuşturılan *VotingContract*, önerilerin sunulması için bir arayüz işlevi görür. Her öneri için bir *ProposalContract* oluşturmaktan ve önerilerin oylama sonuçlarını blokzincir platformuna uygulamaktan sorumlu olan *Singleton Factory* kontrattır.



Şekil 7.3 : Önerilen Düğüm mimarisinin basitleştirilmiş bileşen diyagramı.

DIDRegistryContract, Kurucuları (Founders) ve İlk Geliştiricileri (Initial Core Developers), DDO'ları ile genesis bloğunda sisteme kaydettirmiş olur. Bu kontrat, yönetişime katılan aktörlerin kimlikleri için bilgi deposu olarak kullanılır. Bir ProposalContract, karar alma akışını yönetir ve değişiklik önerisi ile ilgili tüm oylama prosedürlerini içerir. Alınan kararlar VotingContract'a bildirilir. Bunlara ek olarak, yönetim politikaları ile yönetilmesi istenen herhangi bir başka akıllı kontrat, (Şekil 7.3'te X Sözleşmesi olarak gösterilmiştir), Bölüm 7.4.3 'te açıklandığı gibi, PEP arayüzünü gerçekleştirmelidir. DIDRegistryContract ve VotingContract'ların sözde kodları Çizelge B.1'de verilmiştir. ProposalContract için de benzer tanım Çizelge B.2'de verilmiştir.

Bir öneri yaşam çevrimi içinde, sistem bileşenlerinin etkileşimi Şekil 7.4'de gösterilmiştir.



Her oylama, ya belirli bir süre sonra ya da hedef katılım oranına ulaşıldığında sona erer. Katılım oranı, kayıtlı DID'lerin sayısından kolayca hesaplanabilir. Oylama sonunda `ProposalContract` salt okunur olarak mühürlenir ve sonuç Evet ise, `VotingContract`, değişikliği blokzincir'e uygulamak için gerekli adımları yürütür. Böylece, hem gömülü PEP'e ilgili politika gönderilir hem de akıllı kontratın *Policies* isimli durum değişkeni güncellenir. *Policies* değeri, blokzincirin mutabakat mekanizmasıyla korunduğundan, dürüst olmayan düğümlere karşı güvenlik sağlanır.

Bir politikanın yaşam döngüsünün aşamaları aşağıdaki maddelerde açıklanmıştır:

- 216

önlemek için, öneriler ancak mevcut politikalarla çakışmaya sebep olmaması durumunda işleme konulurlar.

- Politikayı uygulamaya sokma, uygulamadan kaldırma. VM, akıllı sözleşmelerin yürütülmesinin deterministik zamanda tamamlanmasını ve sadece izin verilmiş komutlar cinsinden işlemler içermesini garanti altına almak için, akıllı sözleşmeleri yalıtılmış olarak çalıştıran (sandbox) bir ortamdır. Modelimizdeki yönetim akıllı kontratlarının, blokzincir düğümünün protokol katmanlarıyla etkileşim kurması gerekmektedir. Kabul edilen önerilere ait politikaların VotingContract tarafından blokzincir'e otomatik olarak uygulanması, bu erişimin sebeplerinden birisidir. Bizim modelimizde teklifler, karmaşık kodlar yerine yalnızca politikaları içerdiğinden, ön teste gerek yoktur. VotingContract, politika değişikliklerini doğrudan blokzincire uygulayabilir. Ancak bu işlem, VM dışında yaşayan PEP'e ulaşmayı gerektirir. Özel bir işlem koduyla ilişkilendirilmiş komut vasıtasıyla aktif politika kümesini değiştirerek bu işlemi gerçekleştiririz. Bu ayrıcalıklı komut, yalnızca VotingContract tarafından tetiklenebilir. Aynı amaç için blokzincir protokolüne özel bir işlem (transaction) türü eklemek de güvenli ama daha karmaşık bir diğer seçenektir.
- Politika Yürütme. PEP mantığı, Şekil 7.3'te gösterildiği gibi düğümün protokol katmanına yerleştirilmiştir. Başka bir yol da PEP'i akıllı bir sözleşme olarak uygulamak olabilirdi. Ancak, bu yaklaşımın çalışması daha zor olacaktır. Çünkü PEP ile düğümün bileşenleri arasındaki etkileşimlerin de ayrıcalıklı çağrılar yoluyla gerçekleştirilmesi gerekecektir. Tercihimiz sayesinde PEP, protokol katmanı bileşenlerine doğrudan erişebilir durumdadır. Zaten yönetim konularının büyük çoğunluğu, protokol katmanı bileşenlerini ilgilendirir. Akıllı sözleşmeler ise genellikle merkezi olmayan uygulamaların (DApp'ler) iş mantığını içerse de bazı istenirse yönetim sözleşmelerimiz gibi politikalar aracılığıyla yönetilebilirler. Akıllı sözleşmelerin PEP ile etkileşime girmesi için iki yönlü bir API kullanılır. Bu PEP Arayüzü API'si NMI API olarak tanımlanmıştır ve yönetilen nesneler (yani akıllı sözleşmeler için) için Bölüm 3.2.3.4'de açıklanan işlemlere sahiptir.

Modelimizde, herhangi bir akıllı sözleşmenin türeyeceği bir üst sözleşme türü (PBM Entegre Kontrat) bu NMI API arayüzleri içerir. Bu NMI arayüzü ayrıca, PEP'in,

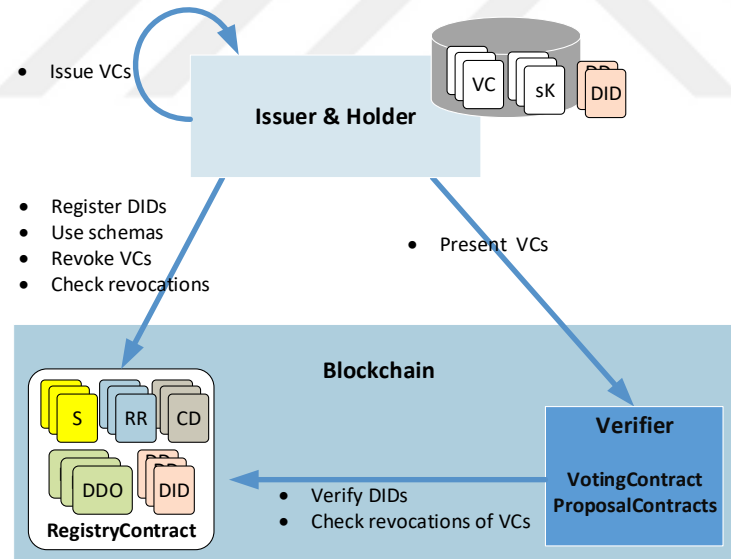
ayrıcalıklı fonksiyonlara ihtiyaç duymadan, düğümün protokol seviyesindeki, yönetilen bileşenlerine erişebilmesine olanak sağlar. Gerçek dünya verileri, yönetilen nesnelerin nitelikleri ve zaman bilgisinin yanı sıra politikalarda kullanılabilir. Akıllı sözleşmeler için mevcut olmayan çeşitli zincirde ve dışında depolanmış veriler de Oracle mekanizmaları sayesinde politikalarda kullanılabilir [256].

7.4.4 DID tabanlı oylama şeması

Bu alt bölüm, tasarladığımız karar alma süreci ile ilgili açıklayıcı bilgiler içermektedir.

7.4.4.1 DID mimarisi

Kurucular, ilk geliştiriciler, geliştiriciler, kullanıcılar, madenciler ve borsa sahipleri olmak üzere altı tür aktör vardır. Modelimizde, kayıt defteri ve aktörlerin sağladığı kanıtlara dayanarak aktörün rolü doğrulanabilir. Basitlik açısından DID ile ilgili verileri ve yönetim prosedürlerini bir IdM blokzincirinde depolamak yerine, yönetilen blokzincire yerleştirmek tercih edilmiştir. Bu durumda, Şekil 7.1’de belirtilen DID referans mimarisi, Şekil 7.5’deki gibi gerçekleşmiştir. Aktörler Issuer ve Holder



Şekil 7.5 : Modelimizde kullandığımız DID Mimari yaklaşımı.

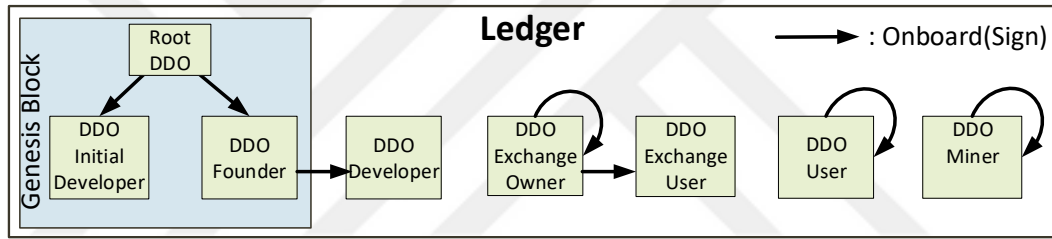
rollerini oynayabilirken blokzincirin kendisi (akıllı kontratlar) Verifier rolünü oynar. Basitlik için sabit Şemalar ve Sertifika Tanımları kullanılmıştır.

7.4.4.2 Aktörlerin sisteme kaydı

Sisteme kayıt (onboarding), aktörlerin DDO'ları aracılığıyla blokzincir'e kayıt olma sürecini ifade eder. Yönetişim sürecine katılmak için, aktörlerin yönetim akıllı

sözleşmeleri tarafından doğrulanabilir olması gerekir (yani, blokzincirde depolanan geçerli DDO'lara sahip olmaları gerekir). İyi yönlendirilmiş oylama sonuçlarına ulaşmak için yönetim ekosistemindeki maksimum katılımcı sayısını bilmek ve yeterli katılım oranını hesaplamak mümkün olur.

Her DID, bir anahtar çiftine karşı düşer ve bu anahtarlar Bölüm 7.3.2 'de açıklandığı gibi yönetilir. Aktörler, üçüncü taraflara güvenmeden kendileri veya delegeleri için kendi anahtarlarını ve diğer parametreleri üretirler. DDO'lar, aktör türüne göre farklı şekilde oluşturulur. Örneğin blokzincir, Kurucuların ve İlk Geliştiricilerin DDO'larını içeren bir genesis bloğu ile başlar. Projeye daha sonra katılan Geliştiricilerin DID değerleri, kayıtlı Kurucular tarafından sisteme kaydedilebilir. Her bir Kurucu, değeri yönetim modelince belirlenen sayıda Geliştirici'yi sisteme kaydettirebilir. Şekil 7.6'da döngülü oklarla gösterildiği gibi, Madenciler, Kullanıcılar ve Borsa Sahipleri (EO'lar) kendi kendilerini sisteme kaydettirebilirler. EO'lar tarafından işletilen



Şekil 7.6 : Aktörlerin kayıt ettirme (güven) bağımlılıkları.

hesapları olan Exchange Kullanıcıları, yönetim süreçlerine katılım için delegasyon talep ettikleri takdirde, EO'ları tarafından sisteme kayıt ettirilebilirler.

7.4.4.3 Oylama süreçleri

Oylamada, aktörler, kimliğini, tercihini (kabul/red/çekimser) ve rolüne bağlı olarak belirli bir Doğrulanabilir Sertifika kümesini (VCs) sağlamak zorundadır. Oylayıcının rolü, sağlanan bu DID ve VC kümesi kullanılarak akıllı kontratlarca (ProposalContract) doğrulanabilir. VC kümesi, sadece rolü değil hesap bakiyesi ve delegasyonları da ispatlayabilir. Sağlanan VC kümesinin doğrulanmasında blokzincir kayıt defterindeki veriler ve RegistryContract içinde depolanmış veriler kullanılır. Oylama sırasında sağlanması gereken VC türleri aktör tipine göre aşağıda listelenmiştir:

- Kurucular ve İlk Geliştiricilerin, genesis blok içinde sisteme kayıt edildikleri için özel anahtarlarının kontrolü haricinde başka bir bilgi ispatlamalarına gerek

yoktur. Kurucular tarafından sonradan kayıt ettirilmiş olan Geliştiriciler ise rollerini ispatlayan VC sağlamak zorundadır.

- Madenciler, rollerini belirten VC'nin yanında, geçerli en az bir coinbase hesabına (sistem tarafından blok ödülü yatırılmış hesap) sahip olduklarını ispatlamak zorundadırlar.
- Kendi hesaplarını işleten normal kullanıcılar, rollerinin yanında hesaplarının sahipliğini de ispatlarlar.
- Kripto borsası kullanıcıları, ancak Borsa sahibinden delegasyon alırlarsa oy kullanabilirler. Teorik olarak Borsa Sahipleri, hesaplarını işlettikleri müşterilerinin bakiyelerini de (delegasyon talep etmemiş olanları) kendi oylarında kullanabilirler.
- Borsa sahipleri ise delegasyon verdikleri müşteri hesapları haricindeki hesapların bakiyelerini ve en az değeri yönetimde belirlenmiş sayıda ve toplam bakiyede hesap işlettiklerini ispatlarlar. Zira, yönetim kontratları, sadece kayıt defterine bakarak, borsadaki müşteri hesaplarını normal kullanıcı hesaplarından ayırt edemezler.

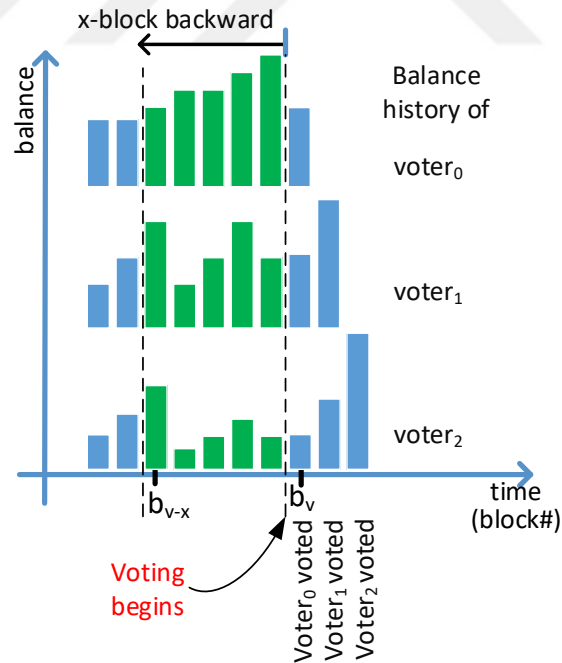
Bu mimari üzerinde doğrudan, temsili ve likit demokrasi gibi çeşitli şemaları inşa edilebilir. Temsili ve likit demokrasi [255], oylayıcıların oy haklarını temsilcilere delege ettikleri demokrasi formatlarıdır. Aktörler kendileri oy verebileceği gibi daha doğru karar vereceklerini düşündükleri üçüncü taraflara oy haklarını devredebilirler. Modelimizde delegasyon şeması, aşağıdaki iki şekilde desteklenmektedir:

- Doğrudan delegasyon şeması: Aktörler oy haklarını, oylama başına olmak üzere, delegasyon sertifikası yayınlamak suretiyle diğer aktörlere tamamen veya kısmen delege edebilirler. Kripto varlıkların sahipliğini transfer etmeden delegasyon mümkündür. Delegasyon yapan aktörler isterlerse delegelerin oylarını kabul etmeyip doğrudan kendi oyları kullanabilirler (liquid democracy).
- Dolaylı delegasyon şeması: Aktörler uzmanların kararlarını kopyalamayı tercih edebilirler. Modelimizde bu amaçla otomatik oy kopyalama (Follow) desteklenir. Aktörler bu işlemi, ilgili ProposalContract'a, kendi DID'sini ve VC kümelerini vererek başlatabilirler. Doğru uzmanı seçebilmek için, geçmiş oylamalarda en çok takipçi almış uzmanları, RegistryContract içindeki

reputation değerlerine bakarak belirleyebilirler. Bir uzmanın *reputation* değeri, kaç kere ve ne kadar büyüklükte kripto varlık delegasyonu ile takip edildiği ile doğru orantılıdır.

Modelde halihazırda öneri başına ayrı ayrı delegasyonlar tanımlama mümkündür. Ama, genel ve/veya yönetim konusuna özel delegasyon tanımlama desteğini de eklemek çok az değişiklik ile kolayca yapılabilir. Ayrıca, birden fazla role sahip bir aktör, farklı DID'ler ile her rolü için bağımsız oylar kullanabilir.

Modelimiz, oy değerlendirmesinde kullanılan kripto varlık bakiyeleri sayesinde hisse bazlı oy verme özelliği içerir. Çöp öneriler ve oylarla sistemi meşgul etmeye karşı bir önlem olarak, diğer yönetim modellerinde genellikle hisse kilitleme yaklaşımı kullanılır. Hisse tabanlı oylama yaklaşımı, aynı zamanda blokzincirlerin *one-user one vote* prensibini garanti edememesi nedeniyle de gereklidir. Çünkü paydaşların kimliklerini belirleyerek çalışmak üzere tasarlanmamışlardır [217]. Modelimizde sunduğumuz bir diğer yenilik, oylama için hisselerin kilitlenmesi ihtiyacını ortadan kaldırmaktır. Uygulanması daha kolay olan ve aynı amaca hizmet eden, *nitelikli bakiye* (eligible balance) olarak adlandırılan yeni bir şema içerir (Şekil 7.7).



Şekil 7.7 : Nitelikli bakiye hesabı.

Nitelikli bakiye, oylamanın başladığı blok numarasından geriye doğru belli sayıda blok öncesinden başlayarak hesaplanan ortalama bakiye değeridir. Yönetim modülleri tarafından otomatik olarak hesaplanır. Sadece sınırlı bir zaman aralığındaki

bakiye değerlerinin kullanımına izin vermek suretiyle sisteme yeni girmiş aktörlerin veya bakiyeleri çok sık değişmiş (shadow voting açısından) riskli aktörlerin karar alma süreçlerine etkileri önlenmiş olur. Böylece muhtemel manipülasyonlar önlenmiş olur.

Üstelik, bir aktör, tekrar tekrar kripto varlık kilitlemesi yapmadan istediği kadar oylamaya katılabilir. *ProposalContract*'daki kontrol mekanizması sayesinde, *bir kullanıcı bir oy* ilkesi asla ihlal edilemez. Ancak, Sybil önerileri ve DoS saldırılarını önlemek için ek bir güvenlik mekanizması olarak *VotingContract* sözleşmesi, öneri sahibinin hesabından, iade etmek üzere belli miktar kripto varlığı güvenlik teminatı olarak kilitler. Kilitlenecek miktarın kendisi de bir yönetim konusudur.

7.4.4.4 Oy verisinin detayları

Çizelge 7.3, *ProposalContract* içindeki oylama süreçlerinde kullanılan DDO ve VC nesnelere ilişkin kriptografik detayları özetler.

Çizelge 7.3 : Oylamada kullanılan DDO ve VC değerleri.

Aktör Tipi	Gereken DDO'lar	Vote Fonksiyonu Girdileri	
	DDO (imza anahtarı ve DID açık anahtarı)	Zorunlu VC'ler VC _{Type} (yayımlayan özel anahtar)	Opsiyonel VC'ler VC _{Type} (yayımlayan özel anahtar)
Founder	DDO _F ($sK_{root}, pK_{founder}$)	None	VC _{Delegation} ($sK_{delegator}$) VC _{AccountPossession} ($sK_{founder}$)
Initial Developer	DDO _{ID} ($sK_{root}, pK_{initDev}$)	None	VC _{Delegation} ($sK_{delegator}$) VC _{AccountPossession} ($sK_{initDev}$)
Developer	DDO _D ($sK_{founder}, pK_{developer}$)	None	VC _{Delegation} ($sK_{delegator}$)
Miner	DDO _M (sK_{miner}, pK_{miner})	VC _{Role} (sK_{miner}), VC _{AccountPossession} (sK_{miner})	VC _{Delegation} ($sK_{delegator}$)
User	DDO _U (sK_{user}, pK_{user})	VC _{Role} (sK_{user}), VC _{AccountPossession} (sK_{user})	VC _{Delegation} ($sK_{delegator}$)
Exchange Owner	DDO _{EO} (sK_{EO}, pK_{EO})	VC _{Role} (sK_{EO}), VC _{AccountPossession} (sK_{EO})	VC _{Delegation} ($sK_{delegator}$)
Exchange User	DDO _{EU} (sK_{EO}, pK_{EUser})	VC _{Role} (sK_{EO}), VC _{AccountPossession} (sK_{EO}), VC _{Delegation} (sK_{EO})	VC _{Delegation} ($sK_{delegator}$)

Bir aktörün DID değerini de içeren DDO bilgisi formül 7.1'de tanımlanmıştır. Bu formülde pK_i , bu aktörün kimlik açık anahtarı ve $signature_{sK}$ ise bu aktörü sisteme kaydettiren aktörün (kendisi veya başkası) kimlik özel anahtarı (sK_i) kullanılarak DDO_i gövdesine atılmış imzadır.

$$DDO_{DID}(sK, pK) = \{DID; pK; other\_data; signature_{sK}\} \quad (7.1)$$

Çizelge 7.3'ün ikinci sütunu, formül(1)'de her bir aktör tipinin kullandığı anahtarları göstermektedir. Bir account-possession-credential ($VC_{AccountPossession}$) bu sertifikanın

yayımcısının, sertifika içinde listelenmiş bütün blokzincir hesaplarının kontrolüne sahip olduğunu ispatlar. Her bir hesap için, sadece o hesabın özel anahtarı ile üretilebilecek olan kriptografik bir hesap kanıtın (yani, hesabın sahipliği) içerir. VC'nin kendisi formül 7.2'de gösterildiği üzere bu aktörün kimlik özel anahtarı ile imzalanmıştır. Formülde 7.3'de açıklandığı gibi AP: *account proof* ve *sig* ise imza bilgisidir (3).

$$VC_{AccountPossession_{DID}} = \{\{AP\}_{j=1}^n; sig_{DID}\} \quad (7.2)$$

$$AP_j = Sign_{ska_j}(accountAddr_j); sig_{DID} = Sign_{SK_{DID}}(encoded VC) \quad (7.3)$$

Formül 7.3'de, ska_j , $account_j$ 'ye ait özel anahtar ve SK_{DID} ise DID ile tanımlı olan ve bu sertifikayı yayımlayan aktöre ait kimlik özel anahtarıdır. Delegation credential ($VC_{Delegation}$) ise yetki devri yapan aktörün DID değerini içeren ek bir alan haricinde aynı formata sahiptir.

Oylamalarda kullanılan sertifikalar (VCs) blokzincire kaydedildikleri için herkesçe görülebilir. Kötü niyetli aktörler bu bilgileri saldırılarında kullanmayı deneyebilirler. Bu bilgilerin saldırılarda yeniden kullanımını engellemek için oylamadaki bütün VC'leri ilgili oylama numarasına bağlantılandırılır. Böylece, DID aktörü tarafından $proposal_k$ için gönderilmiş bir oy formül 7.4'deki gibidir. Formülde $vote \in \{yes, no, abstain\}$ ve sig_i ise aktör tarafından, oy içindeki bütün bilgilerin birleşimine atılmış imzayı gösterir.

$$Vote_{DID}^{proposal_k} = \{vote; proposal_k; \{VC\}_{j=1}^n; binding; sig_{DID}\} \quad (7.4)$$

$$binding = Hash(k || \{VC\}_{j=1}^n)$$

Oylama sonucu formül 7.5'deki üçlü bilgi olarak hesaplanır.

$$r = \sum_{i=1}^{\omega} f(vote_i, role_i, balance_i) \quad (7.5)$$

Bu formülde ω toplam oy veren aktör sayısını, $balance_i$, $actor_i$ 'ye ait nitelikli bakiye miktarını, $vote_i \in \{yes, no, abstain\}$, r ise $(r_{yes}, r_{no}, r_{abstain})$ 'dir. f fonksiyonu, formül 7.6'daki gibidir.

$$f(vote, role, balance) = \begin{cases} r_{vote} += coeff_{role}, & role \in \{founder, initial developer, developer\} \\ r_{vote} += balance \cdot coeff_{role}, & role \in \{user, exchange owner, miner, exchange user\} \end{cases} \quad (7.6)$$

Her aktör türünün, yine bir yönetim konusu olarak belirlenen, farklı bir oy ağırlık katsayısı vardır. Daha çok aktör tipi, yönetim sürecinde daha yüksek çözünürlük ve daha ince ayar yeteneği demektir. İlgili katsayı, yönetim süreçleri yoluyla

sıfırlanarak, herhangi bir rol oylama sürecinin dışında tutulabilir ve bazıları baskın hale getirilebilir. Ayrıca formül 7.5’de, oylama konusuna (protokol, iş, yönetim, arayüz, vb.) göre farklı katsayılar kullanılarak model daha da geliştirilebilir.

7.5 Gerçekleme ve Deneysel Sonuçlar

DID modüllerini uygulamak için, uPort ve kimlik ERC'leri gibi pek çok akıllı sözleşme tabanlı DID araçlarını analiz edilmiştir. Bunların yetenekleri ihtiyacı karşılamadığı için kendi akıllı sözleşmelerimiz geliştirilmiştir. PoC geliştirmemiz için blokzincir platformu olarak Ethereum'u ve Ethereum düğüm yazılımı olarak Hyperledger BESU seçilmiştir. BESU Java tabanlı bir açık kaynak Ethereum istemci uygulamasıdır ve Police PBM bileşenimizle entegrasyonu kolaylaştırmaktadır. Ancak BESU EVM’in mevcut özelliklerinin yanı sıra, aktörler tarafından sağlanan kimlik bilgilerini doğrulayabilmek ve politikaları uygulayabilmek için akıllı sözleşmelerimiz aşağıdaki ek işlevlere ihtiyaç duyar:

- Politika etkinleştirme veya devre dışı bırakma için, akıllı sözleşmeler ile PEP arasında PEP yönetim arayüzü API'sinden kaynaklanan ayrıcalıklı fonksiyonlar.
- Belirli bir hesap adresinin bir madencinin coinbase adresi olup olmadığını kontrol etme yeteneği.
- Nitelikli bakiye hesabı için belirli bir bloktaki bir hesabın bakiyesini almak: Halen akıllı sözleşmelerin ayrıntılı defter verilerine erişim sınırlıdır. Yalnızca en son 256 blok başlığına erişilebilir. İşlemler de erişilebilir durumda değildir.
- Legal öneri sahiplerine, önerileri sırasında harcadıkları maliyeti (ve ödülü) geri ödemek için bir hazine mekanizması kurulmalıdır.
- Devam eden oylamayı sabit bir zamanda veya ulaşılan belirli bir katılım oranında sonlandırmak için bir mekanizma: Ethereum tasarımı herhangi bir dahili programlanmış tetikleyici içermemektedir.
- Ayrıca, DDO'lar ve VC'ler, W3C tarafından JSON Nesneleri olarak tanımlandığından, yönetim akıllı sözleşmelerinin, JSON nesnelerini işleyebilecek, karmaşık ve yüksek maliyetli sözleşme kodları içermesi gerekir.

Yukarıdaki tüm işlevler, ancak standart BESU Ethereum istemci kodunda değişiklik yapılarak gerçekleştirilebilir. PBM-entegre akıllı sözleşmeler ve VotingContract ile PEP arasındaki NMI arayüzü API'sini gerçekleştirmek için, Solidity derleyicisi ve BESU EVM modülünde değişiklik gerekir.

Önceden derlenmiş kontratlar (precompiled), hesaplama açısından çok pahalıya mal olabilecek işlem-yoğun süreçler için kullanılır. Bu tür kontratlar Ethereum düğümlerinde EVM'in dışında çalışırlar. Yönetişim sözleşmelerimizin kullanımı için, hem belirli bir hesap adresinin türünü öğrenme hem de kayıt defteri verilerine erişim amaçlı kullanılan, önceden derlenmiş akıllı sözleşmeler geliştirilmiştir. Kronolojik olarak eski durum değerlerine erişmenin başka bir yolu, bir arşiv düğümüne bağlı bir Oracle [256] aracılığıyla olacaktır. DDO'ları ve VC'leri JSON nesnelerini ayrıştırmak için önceden derlenmiş sözleşme yaklaşımını da kullandık.

Öneri sahiplerini ve seçmenleri ödüllendirmek için bir mekanizma da bu modele kolayca yerleştirilebilir. Yönetişim modeli kapsamında bir hazine mekanizması geliştirmeye başlanmıştır. Ödeme için, örneğin, yeni paralar üretmek veya platforma özgü yollarla doldurulan bir hazine hesabı kullanmak seçenekleri düşünülebilir.

Sürmekte olan bir oylamayı önceden tanımlı bir tarihte sonlandırmak için, blokzinciri dışında çalışan, belirli tarihte çalışacak şekilde işlem planlamaya izin veren Ethereum Alarm Clock gibi yöntemler bulunmaktadır. Çalıştırılacak işlem emrinin detaylarını sağlayıp, gerekli gaz ücretini de önden ödeyerek belirli tarihte bir işlemi tetiklemek mümkündür. Oylamaları bu servisi kullanarak sonlandırmak herhangi bir sakınca içermemektedir. Bu servisin yanında, yönetim modelinin çağırana ödeme yapacağı bir fonksiyon da aynı iş için kullanılabilir.

Normal olarak, Hyperledger Fabric gibi daha esnek sanal makine sınırlarına sahip, izinli (permissioned) blokzincir platformlarında blokzincir düğümü kodunu değiştirmek, Ethereum gibi açık blokzincirlere göre daha kolay ve daha az problemlidir. Bu nedenle modelimizi izinli bir blokzincir platform aracı üzerinde uygulamak, herhangi bir akademik zorluk içermeyen bir çalışma olacağı için tez çalışmasında kapsama almadığımız bir konu olmuştur.

Basitlik, ölçeklenebilirlik ve performans amaçlarıyla, öneri verilerini ve oylama bilgilerini depolamak üzere, her öneri için ayrı bir kopyası çalışan sözleşme tasarlanmıştır. Böylece, daha kısa ve sade sözleşme kodları elde edilebilmiştir.

Çalışmamızda, BESU Ethereum istemci düğümüne PBM yeteneği kazandırmamız gerekmiştir. Yine bu tez çalışması kapsamında geliştirdiğimiz, esnek entegrasyon ve hızlı kod geliştirmeye imkan sunan POLICE PBM modülleri, blokzincir yönetiminde de kullanılmaya uygundur. POLICE PBM çerçevesi, yönetilen nesnelerle etkileşim için NMI Arayüzü olarak adlandırılan bir soyutlama katmanına sahiptir. NMI şu anda JMS (java mesajlaşma teknolojisi) desteklemektedir. Ancak, Ethereum Node'un bileşenleriyle ara bağlantı için, daha yaygın olan RestAPI desteği eklenmesi gerekmektedir. Akıllı sözleşmelerin NMI etkileşimleri, önceden derlenmiş sözleşmeler (precompiled contracts) olarak tasarlanmıştır. Geliştirilen akıllı sözleşmeler ve Ethereum Node yazılımında yapılan değişiklikler Github üzerinden açık olarak erişilebilir durumdadır [257].

Modelin gerçek dünya uygulama örneklerinde kullanılabilirliğini göstermek için, önce BIP ve EIP'de sunulmuş gerçek değişiklik önerilerine karşı düşen bir dizi örnek politika tanımlanmıştır. (Politika örnekleri Çizelge D.1'de görülebilir).

Yönetişim konularıyla ilgili politikalar önermenin (örnek: oylama türleri için kabul eşiklerinin değiştirilmesi ve yeni aktörler için sınırlamalar) oldukça basit bir işlemdir.

7.6 Saldırgan Tehdit Modeli

Bu bölüm, önce mimarimize karşı bir saldırgan modeli kısaca açıklayıp, ardından sistemimizin bunlara nasıl yanıt verdiğini açıklamaktadır. Rasyonel olarak, modelimizin olası tüm saldırılara karşı güvenli olduğunu iddia etmek doğru olmayacaktır. Bu bölümde sadece ilk bakışta akla gelen ana saldırılar incelenmiştir. Modelimizin kullanıldığı blokzincir platformunun niteliklerine bağlı olarak gelecekte farklı saldırı türleri de bulunabilir. Bu analizinden çıkarılan bazı zayıflıklara karşı, modelimize önlemler de eklenmiştir.

Saldırgan Modeli. Mutabakat senaryolarındaki zayıflıklardan faydalanan saldırıların bu çalışmanın kapsamı dışında olduğunu varsayıyoruz. Blokzincirin kendisinden kaynaklanan zayıflıklar da kapsamımızın dışındadır. Bu kısıtlamanın dışında, saldırganlar, yönetim modelini bozmak veya bundan kar elde etmek için sistemin akışını bozmaya kalkabilirler. Örneğin, bir madenci, yönetim altyapısını kendi çıkarları doğrultusunda değiştirmeye çalışabilir. Diğer kullanıcı türleri ise çifte

oylama, hileli oylama, oran deęiřtirme, oylama sonularının kasıtlı maniplasyonu ve rřvet ile oy satma gibi giriřimler yoluyla bireylerin oylarını etkilemeye alıřabilir.

Saldırı Analizleri. Ynetiřim modeli zerinde yrtlebilecek ařaęıdaki muhtemel saldırı senaryoları (T.i) ve nasıl karřı koyulabileceęi irdelenmiřtir:

T.1. Borsa Sahibi, iřlettięi mřteri hesaplarından, oylama iin sahiplerine delegasyon verdiklerinin bakiyesini de ynetiřim srelerine kendi adına katılmak iin kullanmayı deneyebilir. Eęer delege henz oy kullanmamıř ise ynetiřim modelimiz EO'nun oyunu kabul edecek, delegenin sonradan kullanacaęı oyu ise reddedecektir.

- Bu saldırı, borsa kullanıcısının ProposalContract'a oy verme giriřiminde ortaya ıkacaktır. Bu řekilde tesbit edildięinde veya ktye kullanım raporlaması mekanizması ile tesbit edildięinde cezalandırılabilceęini bilmek aktrleri caydırıcı yntem olarak modelimizde bulunmaktadır.

T.2. İlk Geliřtirici veya Kurucu, kendisine izin verilen sayıdan daha fazla Geliřtiriciyi sisteme kaydetmeye alıřabilir.

- nleyici kontrol, RegistryContract tarafından, aktr kaydettirme sırasında yapılmaktadır. Bu szleřmenin kodunu deęiřtirmeden bu saldırıyı yapmak mmkn deęildir. Szleřmenin kodu ise blokzincir tarafından korunur.

T.3. Benzer olarak, kt niyetli İlk Geliřtirici veya Kurucu, kendi kotası kadar hayali Geliřtiriciyi sisteme kayıt ettirerek onlara ait sahte hesaplarla oylama srelerine mdahil olabilir.

- Bizim modelimiz, ynetiřim modelinde bu Geliřtirici kayıt ettirme kotaları tanımlanmıř ise bu trden bir saldırının tespitinde yetersiz kalmaktadır.

T.4. Bir saldırgan aynı neriye birden fazla kere oy vermeyi deneyebilir.

- Semenlerin DID bilgileri RegistryContract'ta kaydedilir. İlgili ProposalContract, oy veren DID'leri ve hesap adreslerini her oylamada ifte oylamaya karřı kontrol eder. Bu nedenle, aynı DID'ler veya hesaplar ile oklu oylama mmkn deęildir.

T.5. Kt niyetli aktr, blokzincirde bir arızaya neden olacak zararlı bir politika nerebilir.

- Aktrler, kt niyetli nerileri, ařık olmadıklarında veya sistemin semantięinin arkasına gizlenmiř olduklarında gzden kaırabilirler. Ancak

aktörlerin çoğunluğu sorunu fark edemezse hasar meydana gelebilir. Modelimizdeki tek koruma mekanizması, politika çakışma tesbit kontrolüdür. Ancak kötü niyetli politikalar, yönetim mekanizmasının kendisini bozmadığı sürece, hasarlar yeni önerilerle çevik bir şekilde düzeltilebilir.

- T.6. Devreye alınan politika nesnesi, önerideki düz metin halinden farklı olabilir.
- Oylamaya başlamadan önce sistem, düz metin halindeki politikayı derleyerek bu durumun geçerli olmadığından emin olmalıdır. Derleme yeteneği mevcut olduğu için öneri ile birlikte derlenmiş politika nesnesine gerek kalmaz.
- T.7. Madenciler, blokzincirde devreye alınmış politika kümesini kendi lehlerine değiştirmeye çalışabilir.
- Normalde, düğüm sahipleri, kendi düğümlerinde, istedikleri herhangi bir politika kümesini ve düğüm yazılımını çalıştırabilir. Ancak bu kötü niyetli madencilerin, bu farklılaştırılmış düğümleri tarafından oluşturulacak bloklar, mutabakat protokolü sayesinde diğer düğümler tarafından kabul görmeyecektir. Çünkü, diğer düğümler, ancak bir önceki bloktaki durum değerleri ile uyumluluk kontrolünü geçebilen blokları kayıt defterlerine eklerler. Modelimizde, blokzincir legal politika kümesi de bir durum değişkeni olarak yönetildiği için blok uyumluluğu kontrolü kapsamına girecektir.
- T.8. Saldırgan, mevcut devreye alınmış politika kümesi ile çakışan bir politikayı içeren değişiklik önerisinde bulunarak sisteme zarar vermeye çalışabilir.
- Oylamaya başlayabilmek için önerilen politika değişikliğinin mevcut politikalarla uyumluluğu otomatik olarak kontrol edildikten sonra öneri oylama aşamasına geçebilir. Buna ek olarak bu tür, çakışan politika önerilerinde bulunan aktörlere (örneğin öneride kilitlenen kripto varlıklarına el koymak gibi) çeşitli cezalar da uygulanabilir.
- T.9. Saldırgan, alternatif yönetim kontratları çalıştırarak yönetim sürecinin kontrolünü ele geçirmeye çalışabilir.
- Saldırgan insanları aldatarak onlara sahte akıllı sözleşmelerini kullandırırsa bile oylanıp kabul edilecek politikayı gerçek blokzincir üzerinde etkinleştiremez. Çünkü politika devreye alma ve kaldırma yetkisi sadece adresi genesi blok ile kayıt altında olan VotingContract'tır. Bu sözleşme için

muhtemel kod güncellemeleri ise yalnızca bir yönetim oylamasından sonra gerçekleştirilebilir.

- T.10. Daha gelişmiş bir saldırı senaryosu, blokzincir kayıt defterinden öğrenilen geçmiş DID ve VC değerlerini kullanarak başka aktörün yerine oy kullanmaktır.
- Bu saldırı ilk tasarımımda mümkün idi. Ancak, her oy nesnesini ilgili oylama numarasına kriptografik olarak bağlayacak mekanizma eklenerek bu saldırı senaryosu önlenmiş oldu.
- T.11. Kötü niyetli aktör, eskiden verilmiş ama iptal edilmesi unutulmuş yetki devirlerinden (delegation VCs) kendi lehine faydalar çıkarmaya çalışabilir.
- İptal edilmemiş yetkiler nedeniyle olabilecek saldırı senaryosu tesbit edilememiştir. Geçerlilik kontrolleri ve ortak alanlardan (Revocation Registry) sorgulanabilecek yetkilendirme iptal durumları kullanılarak ele alınabilecek bir konudur.
- T.12. Bir grup saldırgan, aynı bakiye için birden çok aktöre delegasyon sertifikaları yayınlarak aynı kripto varlıklarından dolayı birden fazla kere oy vermeye çalışabilir.
- Bu saldırıyı önleme amaçlı kontrol, ProposalContract'ın oy verme fonksiyonu tarafından yapılır. Bu kontrolde, verilen oya ilişkin bakiyelerin, aynı oylamada önceden kullanılıp kullanılmadığı denetlenir.
- T.13. Oylama süreçlerine ait işlem emirlerini (transactions) blokzincir üzerinden sürekli izleyen bir saldırgan, oy verenlerin kimliklerini ve verdikleri oyları öğrenebilir. Bu sebeple aktörlerin mahremiyeti tehlike altında girer.
- Oylar içerisinde, kişisel veriler gönderilmemektedir. Aktörlerin kimlikleri ve oyları üzerinde güçlü bir mahremiyet gereksinimi varsa, akıllı sözleşme içinde oyları işlemek için homomorfik şifreleme gibi kriptografik şemalar kullanılabilir.
- T.14. Sıradan bir kullanıcı, Borsa Sahibiymiş gibi oy kullanmaya çalışabilir. (Aslında bu saldırı başka rolleri taklit etmeye dayalı bütün saldırılar için genelleştirilebilir)
- Bu saldırı ilk tasarımda mümkün idi. Önlem olarak eklenen yeteneğe göre; Borsa Sahiplerinin rollerindeki ehliyetlerini ispat etmek için, en az yönetim

modelinde deęerleri tanımlanmış sayıda hesabın ve toplam miktarda bakiyenin kontrolüne sahip olduklarını ispat etmeleri gerekir.

7.7 Analiz Çalışması -2.1: DAO Yönetişimi İçin Kavramsal Çözüm Önerisi

DAO'lar (Decentralized Autonomous Organizations), akıllı sözleşmeler içine kodlanmış kurallarla temsil edilen ve otonom olarak yürüyen sanal kuruluşlardır [258]. Mevcut DAO'ların da blokzincirlerdekine benzer yönetim sorunları yaşadığı literatürden görülebilir. Bu bölümde, blokzincir platformları için önerdiğimiz benzer yeteneklerle donatılmış bir DAO yönetim çerçevesi geliştirmek için izlenebilecek potansiyel bir yaklaşım önerilmiştir. Bu önerinin öncesinde literatür taraması ile DAO'larla ilgili yönetim zorlukları özetlenmiştir. Mevcut DAO inşaa araçları analiz edilerek bu zorluklara ne kadar cevap verebildikleri incelenmiştir.

7.7.1 DAO yönetim zorlukları

Literatürde, DAO'lar için güvenlik ve yönetim sorunlarına odaklanan çalışmalar bulunmaktadır [210, 259, 260]. Bu çalışmalara dayanarak, blockchain ile ilgili zorlukların çoğunun DAO için de geçerli olduğu kolayca görülebilir. Literatürdeki zorluklardan, Blokzincirler için olanlarla örtüşenler aşağıdakilerdir:

- Paydaşların mutabakatı. Genellikle oylama yoluyla *doğrudan demokrasi* metodu ile çalışır. Ancak oylama süreçleri ile ilgili olarak, oylamaya düşük katılım, rüşvet riski (oyları satın alma), gölge oy (shading vote) ve oy haklarının belli aktörlerde toplanması nedeniyle merkezileşme, oylama vasıtasıyla DAO'nun kendisinden çeşitli çıkarlar elde etme saldırıları gibi endişeler vardır. Aktörler takma isimli olduğu için “*bir kişi-bir oy*” ilkesi [214, 217] kolayca ihlal edilebilir. Yönetişimin büyük bir kısmı zincir üzerinde kontrollü yürütülse de DAO'lar, iletişim, tartışma çözme, yönetim işlevleri ve dış görevler gibi zincir dışı operasyonlara da sahip olabilir. DAO'nun güvenliğinin devamı için, zincir dışındaki eylemlerin kurallara uyumluluğunu, kanıtlarını ve şeffaflığını da sağlamak gereklidir. Binlerce veya milyonlarca katılımcı ölçeğinde kolektif karar verme için verimli ve dayanıklı bir mekanizma gereklidir. Özel kararların alınması ve uyuşmazlık çözümü için merkezi olmayan adalet mekanizmaları işletilmelidir. *Kuadratik oylama* gibi mekanizmalar, hisse bazlı merkezileşmeye karşı önlemlerden birisidir.

- Yönetişim güvenliği. Üyeler ve kiralanan yabancılar da dahil olmak üzere DAO aktörleri birbirlerine güvenmezler. Aktörler takma adlara sahiptir ve bu nedenle gerçek kimlikleri ve nitelikleri açık olmayabilir. Oysa, aktörlerin eylemleri şeffaf ve hesap verebilir olmalıdır. Akıllı sözleşmeler, DAO'yu akamete uğratmak veya kötüye kullanmak için *Sybill* aracılığıyla ele geçirmeyi amaçlayan saldırılara karşı dayanıklı olmalıdır. Kötüye kullanım raporlaması, kara listeye alma ve yasaklama gibi önlemler mevcut olmalıdır. Öznel uyuşmazlıklar için merkezi olmayan çözüm mekanizmaları çalıştırılmalıdır. Bu önlemlere rağmen, ne yazık ki, gölge oy ve rüşvet hala birer tehdittir. Yönetişim güvenliği kimlik seçiminden etkilenebilir. Bazı DAO'lar, katkılarına göre paydaşlara itibar ve güven katsayısı atayan kimlikler kullanır. Güvenlik için token/hisse tabanlı yönetim de kullanılmaktadır. Yasal uyumluluk zorunluluklarından (KYC, AML ve CFT gibi) kaynaklanan kullanıcı tanımlama gereksinimleri olabilir. Öte yandan, anonimlik sağlanamaması, ekonomik saldırılara olanak sağlayabileceği için DAO'lara zarar verebilir [260]. Güvenlikle ilgili bir diğer mekanizma, DAO olgunlaştıkça akıllı sözleşmelerin kodunu değiştirmeye gerek bırakmayan (merkeziyetsizliği destekleyen) *dinamik izin* yönetimidir. Uzun akıllı sözleşmeler, DAO'ya saldırı yüzeyini artırır. Bilgisayar korsanları, akıllı sözleşme hatalarına ve güvenlik açıklarına saldırmak ve bundan yararlanmak için açıklıklar bulabilir. Kod güncelleştirmelerinin güvenliğini sağlamak da zordur. DAO kodlarının, hazır şablon kodlar kullanılarak bazı meta-DAO'lar aracılığıyla oluşturulması ve güncellenmesi başka riskli noktalardır. Ne yazık ki, akıllı sözleşmeler devreye alınmadan önce kodun doğrulanması için formal mekanizmalar eksiktir. Aracı portallar ve hazır kütüphaneler kullanarak işlem yapmak çeşitli güvenlik risklerine kapı açabilir. Kodu değiştirmek külfetli ve pahalıdır, bu nedenle acil durum için hızlı yanıt mümkün olmayabilir. Ayrıca, *TheDAO vakasında* olduğu gibi, blokzincirin kendi yönetim eylemleri de DAO'nun güvenliğini etkileyebilir [261].
- Merkezileşme riski. Blokzincirdekilerin aksine, DAO aktörleri daha küçük bir hedef etrafında toplanan daha bilinçli kişilerdir. Bununla birlikte, merkezileşme riskini artıran faktörler vardır. Örneğin, eksik bilgiye sahip üyelerin karar verme sürecine yeterli katılımının olmaması, DAO'nun birkaç

üye tarafından kolayca domine edilmesine neden olabilir. Yönetim kararlarında ağır basan büyük hisse sahipleri de merkezileşmeye yol açabilir. Aslında, günümüzde DAO'lar tam olarak merkeziyetsiz ve özerk değildir ve bazen bu üyelerin kabul ettiği bir durum olabilir. Çoğu DAO merkezi olarak hayatına başlar, zaman içinde merkezi olmayan bir yapıya evrilebilir [259]. Meta-DAO'lar, kurucular ve yöneticiler gibi gereksiz yere aşırı yetkilendirilmiş aktörler de merkezileşmeye yol açabilir. Başlatıcılarının hızlı bir şekilde DAO'yu ayağa kaldırmak ve ardından DAO üzerinde çeşitli görevleri yerine getirmek için ücret ödeyerek faydalandığı aracı platformlar (meta-DAO'lar) vardır.

- Adalet. Bir DAO ile ilgili alınan kararlar objektif ve DAO'nun hedeflerine uygun olmalıdır. Kripto varlıklar sahiplerinden haksız yere alınamaz olmalıdır. Blokzincir platformlarının aksine, DAO'lar genellikle hatalı işlemler düzeltme talebinde bulunmak için bir sorun çözme mekanizmasına sahiptir. Bazı DAO'lar, öznel çatışmalar için tahkim/yargılama sistemine de sahip olabilir. Tahkim mekanizması, kiralanan dış aktörlerin de haklarını aramasına izin vermelidir. DAO yönetişimine katılmak tüm paydaşlara açık olmalı ve üyelerin kararlar üzerindeki etkisi dijital varlıklarının miktarlarıyla orantılı ve rollerine uygun olmalıdır. Mutabakat süreci, “*bir kişi-bir oy*” ilkesini ihlal etmeden oy verme yetkilerini dengelemelidir.
- Teşviklerin yanlış hizalanması (Çakışan çıkarlar). Çok üyeli DAO'larda, bireyler oy verme yetkilerinin önemsiz olduğunu düşündükleri için aidiyet duygusunu kaybederler. Yönetişim sürecine katılmazlarsa oylamalar yeterince adil olmayabilir [216]. DAO'lar genelde, çekimser kalırlarsa finansal kayıpları olabileceğinin farkında olan bilinçli aktörlere sahiptir.
- Etkinlik. Blokzincir sistemlerinde olduğu gibi birden çok platforma yayılmış yönetişim prosedürleri ve araçları, etkinliği azaltır. Yönetişim kararları hiçbir zaman blokzincir örneğinde olduğu gibi hard fork'a yol açmasa da kodların güncellenmesi, DAO işleyişinde kesintilere ve yeni iş yüklerine neden olabilir. Bu nedenle, çok sık oluşmamalıdır.
- Karmaşıklık. Prosedürlerde karmaşıklık ve çok sayıda araca ve aracıya bağımlılık, DAO operasyonlarının zaman ve kaynak gereksinimlerini artırır ve

katılımı olumsuz yönde etkileyebilir. Bir DAO'nun inşası ve bakımı ile ilgili adımlar karmaşık ve hataya eğilimli olmamalıdır. Bununla beraber, bir DAO ile doğrudan etkileşime girmek ortalama kullanıcılar için zor bir konu olduğu için hala aracı sistemler kullanılabilir.

- Esnek yönetim. DAO işlemleri statik değildir ve genellikle zamanla gelişir. Paralel olarak DAO yönetim modelinin de esnek olması beklenir. Ancak, DAO süreçlerini değiştirmek için gereken akıllı sözleşme kod güncellemeleri nedeniyle bu esnekliğe ulaşmak zordur. Acil durumlarda, kuralları değiştirmek, akıllı sözleşme güncellemelerinin tasarlanması ve gerçekleşmesi nedeniyle hızlı bir şekilde olamaz. Bazı DAO inşaa platformlarında bile, kod güncellemeleri çok sınırlıdır veya izin verilmez. En az değişiklik için bile tüm DAO'nun yeniden dağıtılması gerekebilir. Buna ek olarak, değişiklikler bir kez uygulandıktan sonra işlemi geri almak daha zor olduğu için riskli olabilir. Kusurlularsa, kolayca ve hızlı bir şekilde sömürülebilirler.
- Sorunlu durumlarda yavaş yanıt. Bir kararı karalamak ve DAO koduna uygulamak, özellikle acil durumlar için tolere edilebilir zamanlarda olmalıdır. Yavaş ve kontrollü doğası gereği, DAO'lardaki kritik bir hata zamanında çözülemeyebilir. Üçüncü taraflar tarafından tasarlanıp inşa edilmiş DAO'larda geleneksel roller, sorumluluklar ve hesap verebilirlik işe yaramayabilir. Geliştiricilerin genellikle anonim olduğu açık kaynaklı olarak geliştirilen DAO'lar için durum daha da karmaşıktır. Sorumluluk ve hesap verebilirlik eksiklikleri, özellikle acil bir durumda krize neden olabilir.
- Ana hedefleri sürdürmek. Yönetişim işlemlerini DAO'nun hedefleriyle tutarlı sürdürmek kolay olmayabilir. Çünkü üyeler her şeyi derinlemesine analiz edemeyebilirler. Bunun bir sonucu olarak, DAO'nun hedeflerinden uzaklaşması, geriye dönük olarak ve aralarında uyumsuz karar önerilerinin sunulması durumları oluşabilir. Bu uygunsuzluklar belirlenip çözülmedikçe, gözden kaçabilir ve DAO'yu istenmeyen durumlara sürükleyebilir.

Yukarıdakilere, aşağıdaki DAO'ya özel zorluk sınıfları [259, 261, 262] eklenmiştir:

- Fon yönetimi. Kar amacı güden DAO'lar karmaşık bir token ekonomisine dayanır ve fonla ilgili konuların yönetimi zordur. Temettü dağıtımı, yatırım yönetimi, DAO'yu fesih ve bakım maliyetleri arasındaki denge ve DAO'nun

iflas noktası (işlem maliyetleri artarsa) gibi konular bu kapsamda ele alınmalıdır.

- Yasal zorluklar (Hukuk, Yönetmelikler, Yargı). DAO'lar, farklı düzenlemelerin geçerli olduğu farklı ülkelerden paydaşlara sahip küresel kuruluşlardır. Yakın gelecekte, DAO'ların gerçek dünya süreçleri ile temas ettikleri noktalarda çeşitli mevzuat gerekliliklerini gündeme gelebilir. Ancak, üyelerin uyrukları veya konumları hakkında hiçbir mantıksal yargı tesis edilemez. Buna ek olarak, açık kaynaklı proje olarak geliştirilen DAO'ların geliştiricileri sorumlu tutulamaz. Yasalarla uyumlu, otorite dostu DAO şimdilik ütöpik bir fikirdir.

7.7.2 İlgili çalışmalar

Başlangıçta, DAO'ların manuel olarak kodlandığı için DAO sayısı çok düşüktü. *TheDAO* gib olaylar ortaya çıktıkça, DAO'ların gerçekleştirmesini kolaylaştırmak için bazı açık kaynaklı araçlar ortaya çıkmıştır. En önemlileri Aragon [263], DAOstack [264] ve Colony[265] ve olan bu araçlar, *DsaaS (DAO Software As A Service)* olarak adlandırılırlar. Ortaya çıkışları sayesinde DAO'ların sayısı hızla artmaya başladı. Ancak bunların da hala sorunları bulunmaktadır. İstikrarsızlık, kötü dokümantasyon, standartlar ve birlikte çalışabilirlik gibi sorunları bulunur. Aragon en popüler DSaaS çerçevesi olarak bilinir. Aragon, DAO oluşturma için ön tanımlı şirket türleri için akıllı sözleşme şablonları ve Dapp geliştirme SDK'ları sağlar. Bu çerçeve ile DAO'lar oluşturmak ve oluşturulmuş DAO'ları yönetmek (kod güncelleme, yetki yönetimi vb.) için de bir meta DAO platformu sunar. Aragon ile DAO gerçekleştirilmesi bir dizi zorlu adım içerir. Aragon ayrıca subjektif uyuşmazlık çözümü için bir merkezi olmayan yargı altyapısı sağlar.

Diğerlerinden farklı olarak Colony, üyeleri görevlerini tamamlamaya zorlayan mekanizmalar sağlayarak üyelerin oy kullanma ihtiyacını ortadan kaldırmayı savunur. Bir anlaşmazlık için itiraz gelmedikçe, tüm faaliyetler sadece ilgili aktörlerin onayları ile devam eder. İtibara dayalı bir sistemdir. Colony ekosistemi, Ethereum ana ağı üzerindeki bir meta-DAO'dur. Colony DAO'ları, domain ve yetkiler cinsinden bölümlenir. İzin kontrollü fonksiyonlar, bir hesabın, domain içinde çağırma yetkisine sahip olup olmadığını tanımlayan parametrelere sahiptir. Kendine ait fon ve harcama akışları olabilen Domain'ler, örneğin takımları, departmanları ve projeleri temsil

edebilir. Koloninin üç tür token'ı vardır: ödüller, çalışılan saatler için ödeme ve performansa dayalı bonus. Görevler Koloni'deki iş birimleridir. Bir görevin, yöneticisi, gerçekleştircisi ve değerlendiricisi vardı. Bazı işlemler hala zincir dışı olarak yürütülür. Üye itibar yönetimi (reputation mining), tamamlanan görevlerin kanıtları ve anlaşmazlıklar zincir dışı olarak analiz edilir. Kendi yönetim uzayına sahip olan merkezi bir meta-DAO, hayatta kalmak için diğer Colony DAO'larının faaliyetlerinden ücret toplar.

DAOStack, ise diğer ikisinden daha az olgunlaşmıştır. DAOStack, kullanıcıların kendi DAO'larını oluşturmalarını sağlayan Alchemy adı verilen kullanıcı dostu bir arayüz sağlar. Arc modülü DAO yapı taşları ve bileşenleri sağlar. Merkezi DAO Defteri, her bir DAO'daki modüllerin yönetilmesine yardımcı olur ve blokzincirindeki kimlikler aracılığıyla DApp birlikte çalışabilirliğine olanak sağlar. DAOStack izin mekanizması içermez. Fikir birliği için, DAOStack karar verme süreci tekliflerle başlatılır.

Bir DAO'nun işletilmesi derin blokzincir bilgisi gerektirir. Bu üç DSaaS çerçevesi farklı yöntemler ve kavramlar sağlasa da bunları kullanmak uzmanlık gerektirir.

Çizelge 7.4'de, bir dizi ölçüte göre mevcut DSaaS çerçevelerinin özellikleri açıklanmıştır. Seçtiğimiz kriterler, önerdiğimiz yönetim modelindeki güçlü yönlerimize karşı düşen konulardır.

DSaaS çerçeveleri, mevcut blokzincir yetenekleri ile (takma ad kimlikler ve token hisse) sınırlıdır ve kimlik/güvenlik konuları için ekstra bir çözüm sunmaz. İzin yönetimi katı ve detaysızdır. Her üç araç da güvenlik risklerine açık olan Meta-DAO merkezli bir işleyişe sahiptir. Yasal ve mevzuata uygunluk sorunu da ele alınmamıştır.

Hazır DAO yapıtaşları kullanılarak, kod hataları azaltılmaya çalışılır. Akıllı sözleşmelerin bakımı hala karmaşık bir süreçtir. DAOStack ve Colony yapılandırılabilir DAO şablonları ve prototipleri sunar. Yeni yönetim modellerinin gerçekleşmesi zordur, çünkü özel mekanizmalara ve akıllı sözleşmelere izin vermezler. Öte yandan, Aragon tabanlı DAO'lara bir dizi önceden yüklenmiş modüller sağlanır. Ancak, işlevleri ek Aragon uygulamaları aracılığıyla genişletilebilir. Bu nedenle, Aragon'daki bir akıllı sözleşme değişikliği yeni bir DApp dağıtımını gerektirir.

Çizelge 7.4 : DSaaS çerçeveleri karşılaştırması.

Kriter	DSaaS Çerçevesi		
	Aragon	Colony	DAOStack
Kimlik ve Güvenlik yönetimi	• Takma ad aktörler • Kuruluşlar, sözleşmelerinin tüm kontrolünü ANJ sözleşmesine vererek ağı katılabilirler. • Aragon Network tarafından hata bulana ödül programı • İzin koşul için Oracle kullanımı güvenlik ve merkezileştirme riski	• Takma adlar ve iki tür itibar • Ağ genelinde TokenLocking sözleşmesi • Teklif vermek için %0,1 reput. sahip olmak gerekir. • Yöneticiler için önerge verilemez • İtibar ağırlıklı oylama sayesinde bir adam-bir oy örtülü olarak elde edilir	• Takma adlar ve İtibar (katkılara, oylara ve değerlendirmelere dayalı). • <i>ArcHives</i> kayıt defteri aracılığıyla birlikte çalışabilir DAO'lar • Yanlış davranışların cezası, muhbirlerin ödülü.
İzin yönetimi	• AragonOS Çekirdeği'ndeki merkezi izin kayıt defterindeki • Statik/Dinamik denetimler • Özelleştirilemez lojilk ifadeler aracılığıyla • Sabit parametre türleri • Anlaması zor ve karmaşık • Uygulama koda kazılı • Yalnızca erişim politikaları	• Sabit izin türleri var • Alan Adları İçinde, itibar odaklı kararlar, • Core Colony sözleşmelerine izin tabanlı erişim • İzin fonksiyonlarıExtension adlı sözleşmeler içinde gerçekleşir.	• Statik/değiştirilebilir kısıt • Global kısıtlamalar konsensüsle değiştirilebilir • Controller sözleşmesi kısıtlamaları uygular • İzin mekanizması yerine, DAO eylemlerine iliştilen erişim denetimi kodu (genel kısıtlama sözleşmesinin pre/post koşulları) • Değişiklik, yeni akıllı sözleşme eklettirebilir
Kod yükseltme	• Hazır Aragon modülleri ekleyerek/kaldırarak • Yeni yönetim modeli mümkün. • Yeni sözleşmeler devreye alarak yönetim değiştirilebilir	• Hazır şirket şablonu yok • Yeni yönetim modeli mümkün değil • Yeni kontrat ekleyerek özel karar verme lojigi. • Upgrade modu (durumu/sözleşmeleri düzenlemek için) bir Beyaz Hesap Listesinde yer alanlarca tetiklenebilir ve oylarıyla sonlandırılabilir.	• Hazır şirket şablonu yok • Yeni yönetim modeli mümkün değil • Yönetişim yükseltmeleri mümkündür • DAOstack topluluğu tarafından geliştirilen DAO'ları. • Önceden yüklenmiş (evrensel) şemaları/ kısıtlamaları yükseltme
Konsensüs ve Tahkim	• Öneri yok. Yalnızca belirli üyeler tarafından belirli konularda oylama (itibar/hisse bazlı) • Sadece Aragon Network düzeyinde teklif • Subjektif anlaşmazlıklar için merkezi olmayan mahkemeler	• Uyuşmazlık çözüm oylama (hisse, itibar, hibrit) • Oy yerine görev tabanlı akış • Önerge (parametre değişikliği veya ceza için) veya finansman önerileri oylanır • Popüler olmayan teklifler çok uzun sürer • Gizli oyu destekler	• Önerilerle tetiklenen oylama (yatırım önermek, itibar vermek, sözleşmeleri yükseltmek, yeni şemaları ve kısıtlamaları kaydetmek) • Oylanmış önergeler (hisse ve itibar) • Sabit bir zaman diliminde oy kullanma • Popüler olmayan teklifler elenir

Mutabakat ve sorun çözme için genellikle, hisseye dayalı oylama kullanılır. Ancak, kiralanan dış aktörler, kendi inisiyatifleriyle haklarını savunmak için önerge veremezler. Sadece Aragon'un merkezi olmayan bir mahkeme sistemi bu amaca hizmet edebilir.

DAO'nun evrensellik iddiasına rağmen, DSaaS çerçeveleri yerel veya uluslararası mevzuata uyan DAO'lar geliştirmeyi desteklemezler.

7.7.3 DAO'lar için bir çözüm yaklaşımı

DAO yönetim sorunlarına çözüm sunan, blokzincirler için geliştirdiğimiz modeli baz alan (PBM ve DID tabanlı) bir yaklaşımı bu bölümde tanıtmaktayız. Blokzincirlerle benzer zorluklardan muzdarip olan DAO'lar için bizim modelimizin uygulanması oldukça makuldür ve potansiyel yararları Çizelge 7.5'de özetlenmiştir. DAO aktörlerinin doğru tanımlanması ve oylama süreçleri güvenliği açısından gereklidir. Delegasyon ve DID kavramları kullanılarak daha tutarlı ve esnek bir karar verme süreci elde edilebilir. Her seçmenin rolü hakkında kesin bir bilgi, yönetimde iyi yönlendirilmiş ve daha adil bir demokrasiye imkan veren çok faktörlü uzlaşma yeteneğine ulaşılmasına yardımcı olabilir. Buna ek olarak, DID doğrudan, temsili ve likit demokrasi gibi demokrasinin tüm biçimlerini destekleme potansiyeline sahiptir.

DAO görevleri için sözleşmeli personel kiralama da DID özelliği ile gerçekleştirilebilir. Farklı ülkelerden üyelere farklı düzenlemeler kapsamında izin verilmesi çok zordur. Bununla birlikte, DID, farklı mevzuata tabii olan yeni üyelerin kabulü ile ortaya çıkan her türlü kimlik gereksiniminin dinamik olarak benimsenmesini mümkün kılmaktadır.

Dijital kimliklerin yönetim süreçlerine entegre edilmesi, düzenleyici dostu DAO'lara olanak sağlayabilir. DID tabanlı şemaların yönetim ile birleştirilmesi, doğrulanabilir kimlik bilgileri olarak aktörlerin gerçek dünyadaki niteliklerini kullanma sayesinde hem DAO'nun güvenliğini artırabilir hem de düzenlemelere uyumu kolaylaştırabilir. Erişim denetimi politikalarını zorlamak, DAO'lar için gelişmiş bir izin mekanizması da sağlar. DSaaS çerçeveleri tarafından kullanılan oldukça katı ama genel olmayan izin mekanizmalarının yerine politikalar kullanmak, daha dinamik koşul ifadelerine sahip olabilecek esnek kuralları uygulamaya izin verebilir.

Çizelge 7.5 : DAO Yönetişi için önerilen yaklaşımın katkıları.

Kriter	Eklenen mekanizma	Faydaları
Kimlik ve Güvenlik yönetimi	• DID tabanlı üyelik • Doğrulanabilir kimlik bilgileri tabanlı kimlik doğrulama ve aktörlerin yetkilendirilmesi • DID tabanlı yasaklama	• Esnek, dinamik olarak herhangi bir kimlik gereksinimine uyarlama • Gizlilik korur • Mevzuat uyumluluğu, hesap verebilirlik • Aktörün rol ve nitelikleri hakkında kesin bilgi • Bir kişi-bir oy sağlanır. • DAO oluşturma, bakım ve güvenlik yönetimi için üçüncü taraflara bağımlılık yoktur. • Daha kontrol edilebilir
İzin yönetimi	• DAO'nun durumunu ve aktörlerin özniteliklerini bilen erişim denetimi politikası uygulama. • Paydaşların konsensüsleri ile politikaların değiştirilmesi	• Genel amaçlı izin mekanizmaları • Esnek ve özel izin yönetimi • Kullanıcı dostu • Erişim denetimi için gelişmiş destek • Şeffaf ve hesap verebilir
Yükseltilebilirlik	• PBM ile DAO'nun davranışını gerçekleştirme • Üyelerin oylarına göre politikaları dağıtma ve devre dışı bırakma • DAO'nun herhangi bir faaliyeti sırasında politikaları uygulama	• Çevik ve basit • Kod geliştirme yükünü kaldırır (geliştiriciye bağımlılık, kararlara uyumu sağlama vb.) • Kesinti ve yeniden akıllı sözleşme dağıtım gereksinimi yok • Maliyet etkin, kolay geliştirme ve bakım • Yeni yönetim modeli gerçekleştirilebilir • Sözleşmelerin boyut ve karmaşıklığını azaltır
Konsensüs ve Tahkim	• Önerilerin politika şeklinde gerçekleşmesi • DID tabanlı oylama • Karar verme işlevlerine doğrulanabilir kimlik bilgileri sağlama • Politika çakışma çözümleme araçları • DID destekli doğrudan, temsili ve likit demokrasi desteği • Oy parametreleri için homomorfik şifreleme ve Sıfır Bilgi İspatları	• Anlaşılması kolay • İyi yönlendirilmiş bir karar verme ve uyuşmazlık çözümü için çok faktörlü (roller, hisseler, kimlikler vb.) fikir birliği • Zincir dışı faaliyetlerin kanıtlarını kullanabilir • Çoklu karar sürecinin paralel ve bağımsız işlemesi • Ortaklaşa ve güven gerekmeden karar verme • DAO'nun herhangi bir bileşeni (yönetişimin kendisi dahil) yönetim kapsamındadır • Oyları koruyan gizliliği destekler • Kullanıcı dostu politikalar sayesinde karar sürecine katılmak kolaydır • Esnek demokrasi • Otomatik doğrulanabilir dijital anayasa

Mevcut DAO'lar, üyelerin katkı geçmişine ek olarak yalnızca aktörlerin takma adlarına güvenebilir. DID, aktörlerin gerçek kimlik bilgilerini kullanarak bu sorunun üstesinden gelebilir. Sabit kodlanmış izinler yerine, üyelere ait doğrulanabilir kimlik bilgileri sözleşmelere dahil edilebilir. Kanunlara uygunluk kanıtı, DAO'lara kimlik bilgileri olarak da sağlanabilir. Katkının kalitesine ve miktarına dayalı itibarlara ek olarak, yönetim için gerçek dünyadaki itibarları kullanmak da mümkün olabilir. DID, DAO'nun gerçek dünya işlemlerine olan güveni tesis etmeye yardımcı olabilir ve zincir dışında oluşan olayların kanıtlarını gerçekleyebilir.

DAO'lar genellikle önceden belirlenmiş ve/veya sınırlı sayıda üyeyle çalışsa da, üyelik hizmeti dinamik kabulü desteklemeli. DID bu amacı sağlayabilir.

Anonimlik ise istenen bir özelliktir, ancak örneğin birinin DAO'nun kontrolünü gizlice ele geçirme riski vardır. DID, aktörlerin DAO'nun izin verdiği ölçüde anonim kaldığı yapılandırılabilir anonimliğe de izin verebilir. DID ve Sıfır Bilgi teknolojileri sayesinde, mahremiyet amacıyla DAO üyelerinin kimliklerini ve niteliklerini gizlemeyerek iş yapmak mümkündür.

Yönetişim, bir DAO'yu kendi üyelerince belirlenen hedefler doğrultusunda işlemlerini sağlamakla birlikte, akıllı sözleşmelerde insan karar verme sürecinin karmaşıklığına eşdeğer olarak gerçeklemek mümkün değildir. Bununla birlikte, önerilerin üyelere kolayca anlaşılabilir bir politika biçiminde belirtilmesi, DAO yönetişimini daha etkili ve kullanıcı dostu hale getirir. Bu açıdan, PBM teknolojisi bu işe tam olarak uyacaktır. PBM topluluğu, olası anlamsal bağımlılıkları ve politikalar arasındaki çakışmaları otomatik olarak işlemek için birçok araç sağlayabildiğinden, politikalar DAO'nun otomatik olarak doğrulanabilir dijital anayasaları olarak görülebilir. Buna ek olarak, konsensüs süreci politika formundaki tekliflere dayandığından, yönetim oylama prosedürleri için yüksek katılım değerleri kolaylaştırılabilir.

PBM teknolojisi sözleşmelerin karmaşıklığını ve boyutunu da azaltabilir. Politika tabanlı kodlar daha güvenli ve çevik DAO'lara izin verir. Politikalar aracılığıyla ne kadar çok işlevsellik uygulanırsa, daha kısa ve daha denetlenebilir ve sonuç olarak daha güvenli kod mümkün hale gelir. PBM özelliğini DAO'lara dahil etmek, meta-DAO'lar ve DApp geliştiricileri gibi üçüncü taraflara bağımlılığı da azaltacaktır.

PBM yeteneklerini içeren DAO yönetim yaklaşımı mevcut modellerden daha etkili, daha basit ve daha esnektir. Üyelerin, yönetim prosedürlerinin politikaları için

değişiklik önermelerine izin vermek ve bunları uygulamak bu yaklaşımı esnek hale getirir. PBM özelliği sayesinde DAO akıllı sözleşmeleri, yalnızca politikalar değiştirilerek kesintiye veya kod güncelleme olmadan genişletilebilir. Bu, çoğu yükseltme yükünü ortadan kaldırır ve DAO bakımını kolay bir işlem haline getirir.

Ayrıca, politikalarla yönetim prosedürleri, mevcut katı modellerden, daha kontrol edilebilir ve çeviktir. Bu, maliyetli kod güncellemeleri olmadan çevik karar verme ve değişiklikleri etkinleştirmeyi sağlar. Bu yetenek, üyeleri hata olasılıklarını göz önünde bulundurarak değişikliklerden kaçınmak yerine yönetim modelini yükseltmeye teşvik eder. Benzer şekilde, kritik hataların daha hızlı çözülmesini de sağlar.

7.7.4 Sonuç

Blokzincirin uygulama alanları içinde, insanların merkezi olmayan felsefeye uygun olarak işbirliği yapmalarını sağlamak amacıyla hayatımızı değiştirme potansiyeline sahip DAO'ların, gerçekten merkeziyetsiz ve özerk olmalarını sağlayacak bir yönetim modeline sahip olmaları gerekir. Yaygın olarak kullanılan yönetim modelleri henüz DAO'ların yaygın olarak kabul edilmelerini sağlayacak kadar olgun değildir. DSaaS araçları DAO yönetişiminin geliştirilmesini hızlandırabilir, ancak etkili yönetim modellerine sahip olmak çok daha fazla zaman alacaktır.

Blockchain için önerdiğimiz çözüm, teklifleri belirtme, aktörlerin kimliklerini işleme ve değişikliklerin etkinleştirilme şekli gibi birçok açıdan yenilikçi yaklaşımlar kullanır. Bu nedenle, mevcut DAO mimarileri PBM ve DID teknolojilerinden güç alabilirse, birçok DAO sorunu da çözülebilir. Bu alt bölümdeki bu yol izlenirse, özellikle DAO'ların gerçekleştirilmesine ve bakımına, güvenliğe ve paydaş mutabakatına katkıda bulunabilir. Gelecek işler arasında güvenlik, basitlik, adalet, sağlamlık, kullanım kolaylığı, maliyet etkinliği, esneklik ve mevzuat uyumluluğu hedeflerine sahip bir DAO yönetim modeli geliştirmeye odaklandık. Bildiğimize göre bizim analizimiz, PBM ve DID teknolojilerini DAO yönetim alanına dahil etmeyi amaçlayan ilk girişimdir. Aragon topluluğunun da benzer amaçla *uPort* gibi bir Kimlik Yönetim Sistemi'ne (IdMS) entegre etme planı olmasına rağmen, şu ana kadar herhangi bir ilerleme kaydedilmemiştir.

8. SONUÇLAR VE ÖNERİLER

Bu bölümde, tezde sunulan çalışmanın sağladığı katkılar ve gelişmeye açık özellikler özetlenmiştir. PBM, YBH, Simülasyon ve Blokzincir Yönetimi konularında irdelemelere yer verilmiştir. Gelecekte yapılması planlanan çalışmalar paylaşılmıştır.

8.1 Elde Edilenler

Tezin ana katkısı, merkeziyetsiz hesaplama alt türü dahil olmak üzere YBH sistemlerinin yönetiminde politika tabanlı yönetimin kullanılması konusundaki çalışmalardır. Merkeziyetsiz olmayan YBH türleri için genel bir PDCA simülasyonu geliştirilmiştir. Yönetilen sistemin teknolojisinden bağımsız çakışma çözme ve politika uygulama desteğine, basit bir politika tanımlama dili ve politika uygulama altyapısına sahip olan Police PBM çatısı, geliştirilmiş ve simülasyonlarda kullanılmıştır. Bir Merkeziyetsiz YBH tipi olan Blokzincirlerin yönetiminde PBM kullanımı ise detaylı olarak ele alınmış ve yeni bir on-chain model geliştirilmiştir[12].

Tez çalışmasının ana hedefi olan yüksek başarılı ve merkeziyetsiz hesaplama sistemlerinin yönetim ihtiyaçlarının PBM ile karşılanabilirliğini incelemek için, bu sistemler ve yönetim ihtiyaçları literatürden taranmıştır. Bu ihtiyaçların PBM ile karşılanabilirliğini, YBH sistemlerinin simülasyonları üzerinde gösterilmiştir.

Literatürdeki YBH amaçlı simülatörler incelenmiştir. Bilinen YBH mimarilerini modellemek için evrensel bir YBH sistem modeli tasarlanmış ve bu modelin simülasyonları için HeteroSim [9] isimli simülasyon ortamı geliştirilmiştir.

HeteroSim ile modellediğimiz YBH senaryolarında, Police PBM çatısı ile yönetilebilirlik deneyleri düzenleyebilmek için, HeteroSim adlı özgün platforma, ayrık zamanda, gerçek dünya yazılımlarını da simülasyonlara katabilme yeteneği eklenmiştir. HeteroSim'in bu özelliği sayesinde, çeşitli YBH senaryolarında, simülasyon varlığı olarak kodladığımız nesneleri, Police PBM aracı yardımıyla yönetme olanağı elde edilmiştir.

PBM'in, YBH sistemlerinin yönetimi için kullanılabilirliğini göstermek için hem Police ile tanımlanabilen YBH politikaları gösterilmiş hem de simülasyonlar geliştirilip çalıştırılmıştır. PBM çatısının gerçek YBH sistemleri ile kullanılabilirliğinin kavram ispatı için, TARBİL altyapısının politikalar ile yönetimi incelenmiştir. Bu altyapının bulut ortamına taşınması ve mevcut halinin bir YBH sistemine dönüştürülmesi seçeneklerinin ikisinde de politikaların kullanılabilirliği tasarım seviyesinde gösterilmiştir. Tez süresi içinde pratik olarak gerçekleştirilme fırsatı elde edilememiştir.

Diğer bir kavram ispatı için, Merkeziyetsiz platformlarının politika ile yönetimi çalışmamızda, blokzincir yönetim sorunları analiz edilmiş ve mevcut modellerin bunları ne ölçüde çözebileceği incelenmiştir. Bu sorunların çözümüne katkıda bulunmak için, PBM ve DID teknolojilerine dayanan daha basit, adil, sağlam, kullanımı kolay, daha hızlı, genişletilebilir ve esnek bir zincir-içi (on-chain) yönetim modeli geliştirilmiştir. Model, yönetim önerilerinin belirtilmesi, aktörlerin kimliklerinin kullanımı, karar alma mekanizması ve değişiklikler için etkinleştirme yöntemi gibi birçok açıdan özgün yaklaşımlar içerir. DID tabanlı bir karar alma yaklaşımı ve Politikaların dahil edilmesi, yönetim sürecini çok daha basit ve kullanımı kolay hale getirir. Bu çevik model, yalnızca zincir-dışı yönetim modellerinden daha etkili olmakla kalmaz, aynı zamanda zincir-içi modellerden de daha basit ve daha esnektir. Modelimizin başarımı, eş zamanlı olarak çok sayıda öneri işleme yeteneği sayesinde diğer modellerden daha yüksektir. Sadece önceden belirlenmiş konularda değişiklik önerisi yapılmasına izin veren zincir-içi modellerin aksine, birçok zincir-dışı yönetim modelinde olduğu gibi blokzincir ile ilgili herhangi bir konuda politikalar, dolayısıyla öneriler mümkündür.

Çok faktörlü mutabakat ve delegasyon aracılığıyla, daha tutarlı ve iyi yönlendirilmiş bir karar alma süreci elde edilebilir.

Her bir politika önerisi bağımsız olarak işlenerek ve aktörlerin herhangi bir politika çakışmasına yol açmasını da önleyerek sağlanır.

Politika önerilerinin yönetim prosedürlerini değiştirmesine izin vermek ve bu politikaları yönetim sözleşmeleri için uygulamak, modeli esnek hale getirir. Yalnızca yönetim sözleşmelerini değiştirerek, örneğin yeni oylama şemaları eklenebilir.

Hem düşük seviyeli bileşenleri hem de akıllı sözleşmeleri, blokzincirin işleyişini kesintiye uğratmadan yönetmek için PBM bileşenleri blokzincir düğümlerine başarıyla entegre edilebilmiştir. Modelimiz, herhangi bir hesap tabanlı blokzincire uygulanabilir. Oylama fonksiyonları, politika tabanlı öneri yönetim fonksiyonlarından bağımsız olarak da başka sistemlerde, örneğin off-chain yönetim modellerinde kullanılabilir. Modelimiz, izinli blokzincirler için de kullanılabilir.

Bu model, Ethereum standart yapısında birkaç sapma gerektirse de diğer blokzincirlere uygulanırken bu değişiklikler gerekli olmayabilir. POLICE politika dili ile yönetim hedeflerini tanımlayan politikalar hazırlanabileceği gösterilmiştir.

Modellerimizin karmaşıklığı, mevcut modellerden daha düşüktür. Modelimiz, öneride bulunma, aktörler tarafından değerlendirilmesi ve oylama sonrası aktivasyonu gibi aşamaları tek bir araç olarak, daha basit bir şekilde yerine getirmektedir. Aktörlerin, sonuca, rollerine göre etki edebileceği bir karar alma süreci tasarlanmış ve akıllı sözleşmelerle hayata geçirilmiştir. Sürecin basitliği, yönetim sürecine katılımı artırabilir ve daha doğru ve daha adil kararlar alınmasına yardımcı olabilir.

Geliştirdiğimiz modelin, blokzincirler üzerinde hayata geçirilen merkeziyetsiz DAO (Decentralized Autonomous Organization)'lar için kullanılabilirliği konusunda yürüttüğümüz çalışmada [13], DAO'ların literatürden derlediğimiz sorunlarını çözümünde modelimizin sağlayacağı katkılar tesbit edilmiştir.

8.2 Benzer İşlerle Karşılaştırma

Tez kapsamında, YBH sistemlerinin politika tabanlı olarak yönetilmesi konusunda yürüttüğümüz çalışmaların için geliştirdiğimiz genişleyebilir HeteroSim modeli, simülasyon ve gerçek zaman bileşenlerinin, simülasyonda birlikte kullanımına olanak sağlayan tek simülasyon ortamıdır. Gerçek zaman sistemlerinin kodunda herhangi bir değişikliğe gerek kalmadan iki dünyanın elemanları bir araya getirilebilmektedir.

YBH uygulamalarının simülasyonu konusunda çok sayıda simülasyon aracı geliştirilmiştir. Bunların çoğunluğu YBH alt alanlarına odaklanmışlardır veya spesifik bir çalışma için en az fonksiyonla geliştirilmiş ve sonradan idame ettirilmemişlerdir. SimGrid gibi çok azı, bütün YBH alanlarını kapsamayı hedeflemektedir.

Doğruluk ve ölçeklenebilirlik en önemli tasarım kriterleri olarak ön plana çıkmıştır. Bu amaçla, GES, SimGrid ve GridSim simülatörlerinin karşılaştırılması yapılmıştır [196]. HeteroSim'in bu konudaki performansını incelemek için, bu makaledeki deneyler kendi aracımız ile tekrarlanmıştır. Karşılaştırma sonucuna göre GES en iyi ölçeklenebilen araçtır. HeteroSim, GridSim altyapısı üzerine inşaa edildiği için benzer davranışlar göstermiştir. Bellek kullanımı açısından SimGrid ve GridSim'e göre daha iyi performans göstermiştir. YBH sistemlerinin en yüksek çözünürlükle simülasyonları, (daha çok bileşen tipini simüle edebilen) HeteroSim ile mümkün olmuştur. HeteroSim GridSim'den daha iyi performans göstermiştir. Hatta bazı test değerlerinde SimGrid'den bile daha iyi performans göstermiştir. Artan kullanıcı sayısı ve iş yükü ile birlikte özellikle İstatistiksel hesaplamaların artan yükü nedeniyle SimGrid'in gerisinde kalmıştır. HeteroSim aracımız da benzer olarak Java thread kullanım limitleri ile sınırlıdır. Daha az sayıda iş içeren simülasyon oturumlarında sorunsuz çalışsa da birkaç yüzbin adet işi içeren simülasyonlarda yüksek bilgisayar kaynağı ihtiyacı oluşmakta ve simülasyon yavaşlamaktadır. Aracımızın ölçeklenebilirlik konusunda iyileştirmeye açık noktaları vardır.

SimGrid, farklı uzaylar (örneğin HPC ve Grid Computing) için platform modelleri için geliştirilmiştir. Bu sebeple, HeteroSim ve diğer araçlara göre avantajlar sağlar. SimGrid ise daha sonra yapılan eklemelerle, *Peer-to-peer*, *Volunteer* hesaplama alanlarında [266, 267] ve MPI simülasyonlarında [268] kullanılmaya başlanmıştır.

Öte yandan, Zincir-içi yönetim modelimiz, pek çok açıdan özgün katkıları ile mevcut modellere göre çeşitli üstünlüklere sahiptir. Literatürde örneği görülmeyen şekilde, politikaların blokzincir yönetimi için kullanılması, süreçleri çok daha basit ve kullanımı kolay hale getirir. Bu çevik model, yalnızca zincir-dışı yönetim modellerinden daha etkili olmakla kalmaz, aynı zamanda zincir-içi modellerden de daha basit ve daha esnektir. Modelimizin başarımı, eş zamanlı olarak çok sayıda öneri işleme yeteneği sayesinde diğer modellerden daha yüksektir. Sadece önceden belirlenmiş konularda değişiklik önerisi yapılmasına izin veren zincir-içi modellerin aksine, birçok zincir-dışı yönetim modelinde olduğu gibi blokzincir ile ilgili herhangi bir konuda politikalar, dolayısıyla öneriler mümkündür. Bunu sağlamak için model, daha gelişmiş politika dillerinin kullanılmasına izin vermektedir. Çok faktörlü mutabakat ve delegasyon aracılığıyla, daha tutarlı ve iyi yönlendirilmiş bir karar alma süreci elde edilebilir.

8.3 Gelecekte Yapılabilecekler

Police ve HeteroSim çatıları, aşağıda belirtilen açılardan geliştirilebilir:

- PBM araçları ile yönetilen çoklu uzaylar arasındaki birlikte çalışabilirliği araştırmak; ortak veri biçimleri ile YBH kaynakları için pazarlık protokolleri eklemek. Örneğin, DMTF’ni xmlCIM’ çalışmaları [269] kullanılabilir.
- Grid izleme araçlarının (Ganglia), Police PEP için kullanımı sağlanabilir.
- RMS’lerin birbirlerinin kaynaklarının durumunu daha güncel olarak bilebilmesi için, gözlenecek verinin detayı ve gözlem frekansını çalışma anındaki koşullara göre değiştirebilecek olan adaptif bir gözleme sistemi geliştirmek gerekir.
- HeteroSim’in blokzincir modellemek için geliştirilmesi güncel bir çalışma olacaktır.
- Ağ iletişiminin HeteroSim içinde modellenebilmesi desteğinin geliştirilmesi.
- Java Thread mekanizması kullanılması nedeniyle oluşabilen kısıtlamaları aşmak için SimGrid proses yönetim modeli benzeri yürütme yapısını kullanılabirliği incelenmelidir.
- Simülasyon istatistiklerinin toplanması ve gösterilimi mekanizmasının daha az bellek kullanması sağlanmalıdır (dosya ve gözlem servisleri kullanımı vb).
- Yazılım testi, gerçek dünya yazılımlarını simülasyonda kullanma yeteneği nedeniyle HeteroSim için doğal bir uygulama alanı olabilir. Test senaryosu simülasyon varlıkları içine kodlanır. Bu SE’ler, AL üzerinden test edilecek gerçek yazılımlar ile haberleşirler.
- HeteroSim Uyarlama Katmanı içinde daha fazla tipte iletişim teknolojisine hizmet edebilmek için RMI-IIOP, JDBC, Webservice, RestAPI vb. çok kullanılabilecek teknolojiler için vekil varlıklar gerçeklemek gerekir.
- HeteroSim’in gerçek dünya yazılımlarını kullanmasındaki dual zaman sorununa çözüm geliştirmek gerekir. JiST [161]’te belirtilen *actual time* modelini kullanmak veya tüm simülasyon olaylarının saatini, gerçek dünya saatine dönüştüren bir mekanizma geliştirilmesi düşünülebilir.
- YBH mimarilerinin, politika tabanlı yönetilmesi ile yönetilmemesini karşılaştırmak üzere, bu sistemlerin HeteroSim üzerindeki simülasyonların

sonuçlarını, literatürde bu sistemlerin aynı iş yükü ve benzer koşullar altında politika tabanlı yönetilmediklerindeki sonuçlarla karşılaştırılabilir

- İş sıralanmasında ve kaynak tahsisinde *Evolutionary* ve Genetik Algoritmalar ile Lineer programlama kullanan modeller ile modelimiz karşılaştırılabilir.
- YBH mimarileri ve ilişkili olan simülasyon varlıklarını görsel olarak tasarlamak (etki alanı içi ve dışı) için bir araç geliştirilebilir.
- Literatürde, Gridlerde işleri sisteme göndermek için çoğunlukla GlobusToolkit 2.0 servisleri kullanılır. Police PEP'in politika uygulamada bu Grid servisleri ile entegrasyonu konusu incelenmelidir.
- OpenPBS, HTCondor gibi açık kaynak kodlu bir RMS üzerinde, tanımladığımız PBM uyumlu evrensel YBH modelimiz gerçekleştirilebilir.
- Semantik ilişkilerin, birden fazla Yönetilebilir Nesne arasında tanımlanabilmesine olanak sağlanabilir.
- Police politika semantik analiz kısmında politikaların, SIM'de tanımlı olan, yönetilen sistem bilgi modeli ile tutarlılığı analizi eklenmelidir.
- [109]'de, kaynakların statik özelliklerini içeren "*Construction politikaları*" kullanılarak kaynaklar yürütme anında birleştirilerek karmaşık kaynaklar elde edilir. [109]'deki sistemini güçlendirmek üzere POLICE PBM altyapısı entegre edilebilir.
- Bulut Bilişim, dağıtık kaynak paylaşımını sağlayan yeni teknolojilerdendir. Bu sistemlerde de yönetilebilirlik sorunlarının çözülmesi gerekmektedir. Aşağıdaki konuların listelenmesi gereken konulardan bir kısmı listelenmiştir:
 - Bulut servislerinin yönetimi,
 - Bulutlar arası birlikte çalışabilirlik ve bulut federasyonlarının yönetimi,
 - QoS/QoE ve SLA yönetimi,
 - Güvenlik ve veri mahremiyeti yönetimi.
- Modelimizi genişleterek bulut hesaplama mimarilerinin modellenmesinde ve yönetiminde kullanılması, kapsamlı bir araştırma konusu olabilir. Ayrıca *volunteer computing* ve *peer-to-peer computing* alanlarını da kapsayacak şekilde modelimizin genişletilmesi çalışılabilir.

Blokszincir Yönetişimi modelimiz ile ilgili olarak ise aşağıdaki çalışmalar yapılabilir:

- Geliştirdiğimiz modeli, performans ve ölçeklenebilirlik açılarından deneylerle irdelemek gerekmektedir.
- Düşük katılım eşiklerine karşın, aktörleri doğrudan ilgilendirmeyen konularda olsa bile önerilerde oy kullanmaya teşvik edecek mekanizmalar eklenmelidir.
- Blokzincir yönetim modelimize, kuadratik oylama [254] gibi hisse bazlı oylama şemalarını uyarlamak, başka bir çalışma olabilir. Kuadratik oylama, katılımcıların sadece bir konu lehinde veya aleyhinde ne kadar güçlü hissettiklerini ifade ettiği bir karar alma yöntemidir. Akıllı sözleşmelerimizi değiştirerek daha fazla oylama şeması kolayca gerçekleştirilebilir.
- Oylamada mahremiyet konusu, başka bir önemli çalışma alanı olabilir. Homomorfik şifreleme gibi özel tekniklerle oyların içeriği blokzincir üzerinde görüntülenmeksizin çalışır hale getirilebilir. Katılımcıların oy değerleri, rollerine ilişkin ağırlıklarla hesaplanır. Bu, sadece ifşa edilmemiş kimlik bilgileri ve sıfır bilgi ispatlarıyla başarılabilir.
- Blokzincir yönetim modelimiz, hesap tabanlı blokzincir platformları için tasarlanmıolmasına rağmen, UTXO tabanlı platformlar tarafından da uygulanabilecek özellikler içermektedir. Örneğin, yönetim verilerinin, blokzincir dışında ve yönetime konu olan blokzincire bağlantılandırılarak depolanması ve bir harici DID blokzincirinin yönetime entegre edilerek kullanılması senaryosu hayata geçirilebilir.
- DAO'ların (Decentralized Autonomous Organizations) yönetiminde karşılaşılan ve [13]'de irdelediğimiz sorunların çözümü için önerdiğimiz modeli Aragon'a adapte etmek başka bir verimli çalışma konusu olacaktır. Ortaya çıkacak sonuçlar, DeFi platformlarının (DEX:Decentralized Exchange, Crypto Currencies, DAOs, vb.) yönetiminde de kullanılabilecektir.



KAYNAKLAR

- [1] **Stallings, W.** (1993). SNMP, SNMPv2 and CMIP, p. 376, Addison-Wesley, 1993.
- [2] **International Standard 7498-2**, (1989). Information Processing Systems–OSI-Basic Reference Model-Part2: Security Architecture”, first edition, ISO.
- [3] **Dursun, T., Nasır, D. ve Örencik, B.** (2004). Relation Based Policy Conflict Detection, *Proceedings of the 3rd Asia Pacific Int'l. Symp. on Information Technology (3rd APIS)*, İTÜ, İstanbul, Jan. 13-14, 2004, pp.192-198.
- [4] **Dursun, T. ve Örencik, B.** (2004). POLICE Distributed Conflict Detection Architecture, *Proceedings of the IEEE International Conference on Communications (ICC04)*, Paris, June 20-26, 2004.
- [5] **Dursun, T. ve Örencik, B.** (2003). Dağıtık Politika Çakışması Çözme Yaklaşımı, *1. Türkiye Ulusal Yazılım Sempozyumu (UYMS03)*, Ege Üniversitesi, İzmir, Ekim, 2003.
- [6] **Dursun, T. ve Örencik, B.** (2003). Politika Tabanlı Sistemlerde Yeni Bir Dağıtık Çakışma Sezme Yaklaşımı, *20. Türkiye Ulusal Bilişim Kurultayı (TBK2003)*, Eylül 2-5, 2003, Taksim, İstanbul.
- [7] **Dursun, T. ve Örencik, B.** (2003). POLICE: A Novel Policy Framework, Lecture Notes in Computer Science, LNCS 2869, Nov. 2003, pp. 819-827. https://doi.org/10.1007/978-3-540-39737-3_102.
- [8] **Dursun, T.** (2005). A Generic Policy Conflict Handling Model Relation Based Policy Conflict Detection, Lecture Notes in Computer Science, LNCS 3733 (2005) 193-204. https://doi.org/10.1007/11569596_22.
- [9] **Dursun, T., Dağ, H.** (2009). HeteroSim: Heterogeneous Simulation Framework, The 12th Communications and Network Simulation Symposium (CNS09), San Diego, CA, USA.
- [10] **Dursun, T., Dağ, H.** (2014). A Generic Simulation Model For High Performance Computing Systems, The International Conference on Modeling, Simulation and Visualization Methods (MSV14), 2014, Las Vegas, USA.
- [11] **Dursun, T., Dağ, H.** (2017). A Generic Framework for Building Heterogeneous Simulations of Parallel and Distributed Computing Systems, Arab J Sci Eng, Issue 8, 2017, DOI 10.1007/s13369-017-2497-z, 2017.
- [12] **Dursun, T., Üstündağ, B.B.** (2021). A Novel Framework for Policy Based On-chain Governance of Blockchain Networks, Information Processing and Management Journal, vol 58, issue 4, <https://doi.org/10.1016/j.ipm.2021.102556>.
- [13] **Dursun T., Üstündağ, B.B.** (2021). A Conceptual Solution Proposal for Governance Challenges of DAOs, International Conference on Computer Science, Machine Learning and Big Data, 2021.
- [14] **Flynn, M. J.** (1972). Some Computer Organizations and Their Effectiveness, IEEE Transactions on Computers, C-21 (9): pp.948–960, September 1972

- [15] **Casanova, H., Giersch, A., Suter, F.** (2014). Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms, *Journal of Parallel and Distributed Computing*, Elsevier, 2014, 74 (10), pp.2899-2917.
- [16] **Kaplan, J.A., Nelson, M.L.** (1994). A Comparison of Queueing, Cluster and Distributed Computing Systems, Technical Report, NASA Langley Research Center, June, 1994
- [17] **Gabriel, E., Fagg, G., Bosilca, G., Woodall, T.** (2004). Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation, 11th European PVM/MPI Users' Group Meeting, LNCS, Springer, Vol 3241, 2004, pp. 97–104.
- [18] **Krauter K., Buyya R., Maheswaran, M.** (2002), A Taxonomy ve Survey of Grid Resource Management Systems for Distributed Computing, *Software Practice and Experience*, 32:135–164, Feb.21
- [19] **Foster I., Kesselman, C., Tuecke, S.** (2001). The anatomy of the grid (enabling scalable virtual organization), *Int. Journal Supercomputer Applications*, 15, 2001.
- [20] **Litzkow, M., Livney, M., Mutka, M.** (1988). Condor - a hunter of idle workstations, *8th Intl Conf. on Distributed Computing Systems*, 1988, pp. 104-11.
- [21] **Byun, C., Duncan, C.** (2000). A Comparison of Job Management Systems in Supporting HPC ClusterTools, Presentation for SUPERG, Vancouver, Fall 2000
- [22] **Foster, I., Kesselman, C.** (1999). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.
- [23] **De Jongh, J. F. C. M.** (2002) *Share Scheduling in Distributed Systems*, Delft Technical University.
- [24] **Kumar, K.P., Agarwal, A., Krishnan, R.** (2004). Fuzzy based resource management çatısı for high throughput computing, *IEEE Int. Symposium on Cluster Computing and the Grid*, 2004. CCGrid 2004, 19-22 April 2004 pp:555-562
- [25] **Lifka, D.A.** (1995). The ANL/IBM SP Scheduling System. In D. G. Feitelson and L. Rudolph, editors, *IPPS'95 Workshop: Job Scheduling Strategies for Paralel Processing*, pages 295–303. Springer, Berlin, *Lecture Notes in Computer Science LNCS 949*.
- [26] **I. Foster I., Kesselman, C.** (1997). Globus: A metacomputing infrastructure toolkit, *Intl. J. Supercomputer Applications*, 11(2):115–128, Summer.
- [27] **Weiss, A..** (2007) Computing in the clouds. *NetWorker*, 11(4):16–25, Dec. 2007
- [28] **Calheiros, R. N. Ranjan, R., Beloglazov, A., De Rose, C. A. F., Buyya, R.,** (2011). CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms, *Software: Practice and Experience*, 41 (1), 2011, pp.23–50
- [29] Berkeley Open Infrastructure for Network Computing, Available at <http://boinc.berkeley.edu/>, 2014
- [30] **Yaga, D., Mell, P., Roby, N., Scarfone, K.** (2018). Blockchain Technology Overview, NISTIR 8202, <https://doi.org/10.6028/NIST.IR.8202>
- [31] **Demers, A. Dan, G., and Doug, T.** (1987). Epidemic Algorithms for Replicated Database Maintenance, *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing. PODC '87*, pp. 1–12. doi:10.1145/41840.41841

- [32] **Nakamoto S.** (2008). Bitcoin: A Peer-to-Peer Electronic Cash System, <https://bitcoin.org/bitcoin.pdf>, Aralık 2008
- [33] **Lamport, L., Shostak R. ve Pease, M.** (1982). The Byzantine Generals Problem, *ACM Transactions on Programming Languages and Systems*, V.4, pp. 382-401, July 1982
- [34] **HP-OpenView**, available on <http://www.hp.com/rnd/products/netmgt/netmgt.htm>
- [35] **IBM-NetView**, available on <http://www.raleigh.ibm.com/nd6/ndbhm.htm>.
- [36] **Damianou, N., Dulay, N., Lupu, E., and Sloman, M.** (2000). Ponder: A Language for Specifying Security and Management Policies for Distributed Systems. The Language Specification - Version 2.3. *Research Report DoC 2000/1*, Imperial College of Science Technology and Medicine, Department of Computing, London, 20 October 2000.
- [37] **Alpers, B. and Plansky, H.** (1995). Concepts and application of Policy -based management, in A. S. Sethi, Y. Raynaud and F. Faure-Vincent (eds), *Integrated Network Management IV*, Chapman & Hill, London, pp. 57-68.
- [38] **Masullo, M.J. and Calo, S.B.** (1993). Policy Management: An Architecture and Approach, *Proceedings of IEEE Workshop on Systems Management*, UCLA, California, April 1993.
- [39] **Mahon, H.** (1999). Requirements for a Policy Management System. *IETF Internet draft*, Available from www.ietf.org, 22 October 1999.
- [40] **Rajan R. et al.** (1999). Policy Action Classes for Differentiated Services and Integrated Services, IETF, Internet draft, draft-rajon-policy-qoschema-01.txt, 5 Apr 1999
- [41] **Bernet, Y., Yavatkar R., Ford, P., Baker, et al.** (1998). A Framework for End-to-End QoS Combining RSVP/Intserv and Differentiated Services, IETF <draft-ietf-DiffServ-rsvp-01.txt>, November 1998.
- [42] **M. Fine, K. McCloghrie, S. Hahn, et al.** (1999). An Initial Quality of Service Policy Information Base for COPS-PR Clients and Servers, draft-mfine-cops-pib-02.txt, October 1999.
- [43] **Herzog, S.** (1999). Signaled Preemption Priority Policy Element, IETF <draft-ietf-rap-signaled-priority-04.txt>, September 1999.
- [44] **Lupu, E.** (1998). A Role-Based Framework for Distributed Systems Management. PhD Thesis, Dept. of Computing, Imperial College. London, UK, July 1998.
- [45] **IETF**, Internet Engineering Task Force, Policy Working Group, available online from <http://www.ietf.org/html.charters/policy-charter.html>
- [46] **Strassner, J., Ellesson, E. and Moore, B.** (1999). Eds., Policy Framework Core Information Model, *Internet draft, draft-ietf-policy-core-schema-02.txt*, Feb.
- [47] **OMG**, 1999. Object Management Group, The Common Object Request Broker: Architecture and Specification, Revision 2.3.1, October 1999.
- [48] **Baltatu, M., Lioy, A. and Mazzocchi, D.** (2000). Security Policy System: status and perspective, *IEEE International Conference on Networks (ICON'00)*, 2000
- [49] **Boyle, J., Cohen, R., Durham, D., Herzog, et al.** (1999). The COPS Common Open Policy Service) Protocol, *IETF <draft-ietf-rap-cops-07.txt>*, August 1999.
- [50] **Wahl, M., Howes, T. and Kille, S.** (1997). Lightweight Directory Access Protocol (v3), RFC 2251, available online from [/www.ietf.org](http://www.ietf.org), December 1997.

- [51] **Wies, R.** (1994). Policies in Network and Systems Management-Formal Definition and Architecture, *Journal of Network and Systems Management*, **2(1)**, 63-83, Plenum Press.
- [52] **Dennis K. David B., Balenson, M.** (2000). "Policy-Based Cryptographic Key Management: Experience with the KRP Project," discex, pp.0103, DARPA Information Survivability Conference & Exposition - Volume 1.
- [53] **Moffett, J. D. and Sloman, M. S.** (1993). Policy Hierarchies for Distributed Systems Management. *IEEE JSAC Special Issue on Network Management*, **11(9)**, pp. 1404-1414.
- [54] **Sloman, M. S.** (1994). Policy Driven Management for Distributed Systems, *Journal of Network and Systems Management*, **2(4)**, 333-360.
- [55] **Weis, R.** (1994). Policy Definition and Classification: Aspects, Criteria and Examples. *In Proceedings of the IFIP/IEEE Int. Workshop on Distributed Systems: Operations & Management*, Toulouse, France, 10-12 October 1994.
- [56] **Damianou, N. C.** (2002). A Policy Framework for Management of Distributed Systems, *PhD Thesis, Dept. of Computing, Imperial College*. London, February 2002.
- [57] **Jajodia, S., P. Samarati and V. S. Subrahmanian** (1997). A Logical Language for Expressing Authorisations. *In Proceedings of the IEEE Symposium on Security and Privacy*, pp. 31-42, May 4-7, 1997.
- [58] **Ortalo, R.** (1998). A Flexible Method for Information System Security Policy Specification. *In Proceedings of the 5th European Symposium on Research in Computer Security (ESORICS 98)*, Belgium, Springer-Verlag, September.
- [59] **Lobo, J., Bhatia, R. and Naqvi, S.** (1999). A Policy Description Language. *In Proceedings of the Sixteenth National Conference on Artificial Intelligence Eleventh Innovative Applications of AI Conference*.
- [60] **Stone, G.** (2000). A Path-Based Network Policy Language", Dept. of Comp. Sci., Naval Postgrad. School, Monterey, CA, Sept. 2000.
- [61] **Moore, B., Ellessen, E., Strassner, J. and Westerinen, A.** (2001). Policy Core Information Model - Version 1 Specification, RFC 3060, available from <http://www.ietf.org>, February 2001.
- [62] **Ribeiro, C., Zuquete, A. and Ferreira, P.** (2001). SPL: An access control language for security policies with complex constraints. *In Proceedings of the Network and Distributed System Güvenlik Symposium (NDSS'01)*, San Diego, California, February 2001.
- [63] **Damianou, N. C., Dulay, N., Lupu, E., Sloman, M.** (2000). Managing Security in Object-based Distributed Systems using Ponder. *In Proceedings of the 6th Open European Summer School (Eunice 2000)*, Enchede, The Netherlands, 13-15 September 2000.
- [64] **Damianou, N. C., Dulay, N., Lupu, E., Sloman, M.** (2001). The Ponder Policy Specification Language. *In Proceedings of the Policy Workshop 2001*, HP Labs, Bristol, UK, Springer-Verlag, 29-31 January 2001.
- [65] **Damianou, N. C., T. Tonouchi, Dulay, N., Lupu, E., Sloman, M.** (2002). Tools for Domain-based Policy Management of Distributed Systems. *In Proceedings of the Network Operations and Management Symposium (NOMS 2002)*, Florence, Italy, 15-19 April 2002.

- [66] **Chomicki, J., Lobo, J. and Naqvi, S.** (2000). A Logic Programming Approach to Conflict Resolution in Policy Management, *In Proceedings of the 7th Int. Conference in Principles of Knowledge Representation and Reasoning, Breckenridge, Colorado, USA*, pp. 121-132, April 2000.
- [67] **Hitchens, M. and Varadharajan, V.** (2001). Tower: A Language for Role Based Access Control. In Proceedings of the Policy Workshop 2001, HP Labs, Bristol, UK, Springer-Verlag, 29-31 January 2001.
- [68] **Dulay, N., Lupu, E., Sloman, M. and Damianou, N. C.** (2001). A Policy Deployment Model for the Ponder Language. *In Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management (IM'2001): Integrated Management Strategies for the New Millennium*, Seattle, Washington, USA, 14-18 May 2001.
- [69] **Cengarle, M. V. and Knapp, A.** (2001). On the Expressive Power of Pure OCL. Technical Report 0101, Ludwig-Maximilians-Universität München, Munich, Germany.
- [70] **Z. Fu, S. F. Huang, W.H., Loh, K., Gong, F.** (2001). IPSec/VPN Security Policy: Correctness, Conflict Detection and Resolution, *IEEE Policy 2001 Workshop*, Jan. 2001, Springer 1995, 39-56.
- [71] **Haixin, D., Jianping, W. et al.** (2000). Policy-based Access Control Framework for Large Networks, *IEEE Int. Conference on Networks (ICON'00)*, 2000
- [72] **Verma, D. C.** (2001). Policy-Based Networking: Architecture and Algorithms, New Riders Publishing, 2000.
- [73] **Muruganantha, N. and Lutfiyya, H.** (2003). Policy Specification And Architecture For Quality Of Service Management, *Integrated Management Symposium(IM03)*, 2003
- [74] **Diao, Y., Eskesen, F. et al.** (2003). Generic On-Line Discovery Of Quantitative Models For Service Level Management, *Integrated Management Symposium(IM03)*, 2003.
- [75] **DMTF** official web site is www.dmtf.org
- [76] **Yang, Y., Fu, Z. and Felix Wu, S.** (2003). Management System For *IPSec/VPN*, *Integrated Management Symposium(IM03)*, 2003
- [77] **Waldbusser, S. et al.** (2002). Policy based management mib. Draft-ietf-snmpconf-pm-10.txt, Feb. 2002.
- [78] **Martinez, P. et al.** (2002). Using the Script MIB for Policy-based Configuration Management, *In Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS2002)*, 2002.
- [79] **Tennenhouse, D. L., Smith, J. M., et al.** (1997). A Survey of Active Network Research. *IEEE Communications Magazine*, vol. 35(1), pp. 80-86, 1 January.
- [80] **Fonseca, M., Agoulmine, N., and Cherkaoui, O.** (2001). Active Networks as a Flexible Approach to deploy QoS Policy Based Management, *HP Openview University Association, 8th Annual Workshop, HP-OVUA* <http://www.hpovua.org/>, Berlin, Germany, June 24 – 27, 2001.

- [81] **Herrmann, P., Wiebusch, L. And Krumm, H.** (2002). State-Based Güvenlik Policy Enforcement In Component-Based E-Commerce Applications, *2nd IFIP Conference on E-Commerce*, Lisbon, October 2002.
- [82] **Martinez P. G., Garcia F. J.,** (2007), POSITIF: a policy-based security management system. In Proceedings of the Eighth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'07)
- [83] **Weinberg, N.** (1999). Policy -based networks: Easier said than done, *Network World Magazin*, 08/23/99.
- [84] **Caruso, J.** (1999). Policy -based management ain't what it used to be, *Network WorldMagazin*, available on WWW from <http://www.nwfusion.com/archive/1999b/0412edit.html> 04/12/1999.
- [85] **Gagnon, E.** (1998). SableCC, An Object-Oriented Compiler Framework. MSc Thesis, School of Computer Science, McGill University. Montreal, Canada, March 1998.
- [86] **Lupu, E. C. and Sloman, M. S.** (1999). Conflicts in Policy -Based Distributed Systems Management, *In IEEE Transactions on Software Engineering - Special Issue on Inconsistency Management*, **25(6)**, pp. 852-869.
- [87] **Samarati, P., Vimercati, S.** (2000). Access Control: Policies, Models, and Mechanisms. In Foundations of Security Analysis and Design (*Tutorial Lectures*). R. Focardi and R. Gorrieri eds, Springer -Verlag, pp. 137-196, September 2000.
- [88] **OMG**, 1999, Object Management Group, Object Constraint Language Specification, version 1.3, Chapter 7 in OMG Unified Modelling Language Version 1.3, June 1999.
- [89] **Strassner, J. and Schleimer, S.** (1998). Policy Framework Definition Language (version 00), IETF Internet draft work in progress, available from <http://www.ietf.org>, 17 November 1997.
- [90] **Sloman, M., Twidle, K.** (1994). Domains: A Framework for Structuring Management Policy. *In Chapter 16 in Network and Distributed Systems Management* (Sloman, 1994ed), pp. 433-453.
- [91] **Yialelis, N.** (1996). Domain-Based Security for Distributed Object Systems. PhD Thesis, Department of Computing, Imperial College. London, U. K., August 1996.
- [92] **CIM** (1999). Common Information Model (CIM) Specification version 2.2, *Distributed Management Task Force, Inc. (DMTF)*, available online from www.dmtf.org/spec/cims.html, 14 June 1999.
- [93] **Sun** (1999). Microsistemleri, Inc., Java Naming and Directory Interface, Application Programming Interface, <http://java.sun.com/docs/index.html>, July 1999.
- [94] **Apache Directory Server**, <<http://www.apache.org>>
- [95] **Wahl, M., Howes, T. and Kille, S.,** (1997). Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions, RFC 2252, <http://www.ietf.org>, December 1997.
- [96] **OpenJMS**, available on WWW from <http://openjms.sourceforge.net>.
- [97] **Hapner, Burrridge, Sharma,** (2020). Java™ Message Service API Tutorial and Reference: Messaging for the J2EE™ Platform, Addison-Wesley, Feb 2002
- [98] **Sun WBEM SDK** -<http://wbemservices.sourceforge.net/>

- [99] **BeanShell**, <http://www.beanshell.org>
- [100] **Clark, D., Wilson D.R.** (1987). A Comparison of Commercial and Military Computer Security Policies. In *Proceedings of the IEEE Symposium on Security and Privacy*, 1987.
- [101] **Sanguandikul, N.; Nupairoj, N.** (2006). Implicit information load sharing policy for grid computing environment, Advanced Communication Technology Conference, 2006. ICACT 2006, Volume 3, 20-22 Feb. 2006 Page(s):5 pp
- [102] **Abawajy, J.H.** (2004). Fault-tolerant scheduling policy for grid computing sistemleri, 18th International Paralel and Distributed Processing Symposium, 2004., 26-30 April 2004 Page(s) : 238
- [103] **He, L., Jarvis, S.A., Spooner, D.P., Chen, X., Nudd, G.R.** (2004). Hybrid performance-based workload management for multiclusters and grids, Software, IEE Proceedings- [also Software Engineering, IEE Proceedings], Volume 151, Issue 5, 7 Oct., Page(s):224 – 231.
- [104] **Mohamed, H.H., Epema, D.H.J.** (2004). An evaluation of the close-to-files processor ve data co-allocation policy in multiclusters, IEEE International Conference on Cluster Computing, 2004, 20-23 Sept. 2004 Page(s):287 – 298
- [105] **Wiriayaprasit, S., Muangsin, V.** (2004). The impact of local priority policies on grid scheduling performance and an adaptive policy based grid scheduling algorithm, 7th High Performance Computing and Grid in Asia Pacific Region, 2004, 20-22 July 2004 Page(s):343 – 346
- [106] **Hui Li; Groep, D.; Templon, J.; Wolters, L.** (2004). Predicting job start times on clusters, IEEE International Symposium on Cluster Computing and the Grid, 2004. CCGrid 2004, 19-22 April, Page(s):301–308.
- [107] **Ricardo Neisse, Evandro Della Vecchia Pereira,** (2004). An Hierarchical Policy-Based Architecture for Integrated Management of Grids and Networks, Fifth IEEE Int. Workshop on Policies for Distributed Systems and Networks (POLICY'04)
- [108] **Uszok, A., Bradshaw, J. M., Jeffers, R., Johnson, M., Austin T. Dalton, J.** (2004). Policy and Contract Management for Semantic Web Services, AAAI Spring Symposium on Semantic Web Services, 2004.
- [109] **A. Sahai, S. Singhal, V. Machiraju,** (t.y) Automated Policy-Based Resource Construction in Utility Computing Environments,
- [110] **Graupner, S., Sahai, A.** (2005). Policy-based Resource Topology Design for Enterprise Grids, CCGrid 2005, 9-12 May 2005, Cardiff, United Kingdom
- [111] **Sahai, A., Singhal, S., Machiraju, V.** (t.y) Automated Policy-Based Resource Construction in Utility Computing Environments
- [112] **Verma D.** (2001). Mandis Beigi ve Raymond Jennings, Policy based SLA Management in Enterprise Networks, Policy workshop 2001.
- [113] **In, J., Avery, P., Ranka, S.** (2004). Policy Based Scheduling for Simple QoS in Grid Computing, 2004 IPDPS International Paralel ve Disitributed Processing Symposium Nisan 2004.
- [114] **Sadighi, B.F., Sergot, M.** (2004). A Framework for Contractual Resource Sharing in Coalitions, Fifth IEEE International Workshop on Politicies for Distributed Systems and Networks (POLICY'04).
- [115] **Devarakonda, M., Naik, V.K., Rajamani, N.,** (t.y) Policy-Based Multi-Datacenter Resource Management.

- [116] **Naik, V.K., Zhao, Y.** (2006). On-line Evolutionary Resource Matching for Job Scheduling in Heterojen Grid Environments, 12th International Conference on Paralel and Distributed Systems (ICPADS'06)
- [117] **Jang-uk In, Avery, P., Cavanaugh, R., Chitnis, L., Kulkarni, M., Ranka, S.** (2005). SPHINX: A Fault-Tolerant System for Scheduling in Dynamic Grid Environments, 19th IEEE International Paralel and Distributed Processing Symposium, 2005, 04-08 April 2005, 12-b
- [118] **Dumitrescu, C.L., Michael Wilde, Foster, I.** (2005). A Model for Usage Policy based Resource Allocation in Grids, IEEE Int. Symposium on Cluster Computing and the Grid, CCGrid 2005., Volume 2, 9-12 May, pp 1151 - 1158 Vol. 2.
- [119] **Dumitrescu C.L., Foster, I.** (2004). Usage Policy-based CPU Sharing in Virtual Organizations, Fifth IEEE/ACM International Workshop on Grid Computing, Page(s): 53 – 60.
- [120] **Dumitrescu, C.L., Foster, I.** (2005). GangSim: a simulator for grid scheduling studies, I., IEEE International Symposium on Cluster Computing and the Grid, 2005. CCGrid 2005, Volume 2, 9-12 May 2005 Page(s):1151 - 1158 Vol. 2
- [121] **Otoo, E., Rotem, D.** (2003). Shoshani, Impact of admission and cache replacement policies on response times of jobs on data grids, A., International Workshop on Challenges of Large Applications in Distributed Environments, 2003, 21 June 2003 Page(s):113 - 123
- [122] **Di Martino, V.,** (2003). Sub optimal scheduling in a GRID using genetic algoritmas, International Paralel and Distributed Processing Symposium, 2003. 22-26 April 2003 Page(s):7 pp.
- [123] **Jiang, X., Xu, D.** (2003). SODA: a service-on-demand mimari for application service hosting utility platforms, 12th IEEE Intl. High Performance Distributed Computing, 2003 Symposium, 22-24 June 2003 pp:174 – 183
- [124] **Sanguandikul, N., Nupairoj, N.** (2006). Implicit information load sharing Policy for grid computing environment, Advanced Communication Technology Conference, 2006. ICACT 2006, Volume 3, 20-22 Feb. 2006 Page(s):5 pp
- [125] **Monroy, J., Becerra, J.A.; Bellas, F., Duro, R.J.,** (2006). Paralel Job Scheduling through Evolutionary Based Cognitive Strategies, IEEE Congress on Evolutionary Computation, 2006. CEC 2006, 16-21 July 2006 Page(s):3274 – 3280
- [126] **Xavier, P., Cai, W., Lee, B.** (2006). Adaptive policing for token-exchange based management of shared computing resources, 6th IEEE International Symposium on Cluster Computing and the Grid, 2006. CCGRID 06, Volume 1, 16-19 May 2006 Page(s):8 pp
- [127] **Zeng, Q., Huang, C., Chen, D.; Hunan, H.H.** (2004). Supporting secure collaborative computing in grid environments, The 8th International Conference on Computer Supported Cooperative Work in Design, 2004, Volume 2, 26-28 May 2004 Page(s):413 - 418
- [128] **Thompson, M.R., Essiari, A.** (2003). Fine-Grained Authorization for Job and Resource Management Using Akenti and the Globus Toolkit, CHEP 03, La Jolla, Mar 24-28, 2003,
- [129] **Bertino, E., Mazzoleni, P., Crispo, B., Sivasubramanian, S.,**(2004) Towards supporting fine-grained access control for Grid resources, 10th IEEE International Workshop on Future Trends of Distributed Computing Systems, FTDCS 2004, 26-28 May, Page(s):59 – 65

- [130] **Xavier, P., Lee; B.S., Cai, W.** (2003). A decentralized hierarchical scheduler for a grid-based clearinghouse, International Parallel ve Distributed Processing Symposium, 2003, 22-26 April 2003 Page(s):8 pp.
- [131] **Talwar, V., Basu, S., Kumar, R.,** (2003). An environment for enabling interactive grids, 12th IEEE International Symposium on High Performance Distributed Computing, 2003, 22-24 June 2003 Page(s):184 – 193
- [132] **Neumair, B., Wies, R.** (1996). Case study: applying management policies to manage distributed queuing systems, Distributed System Eng. 3, (1996), 96-103
- [133] **Ranjan, R., Harwood A., Buyya, R.** (t.y) Grid Federation: An Economy Based, Scalable Distributed Resource Management System for Large-Scale Resource Coupling, University of Melbourne.
- [134] **Runsewe, O.A.** (2014). A Policy-Based Management Framework For Cloud Computing Security, University of Ottawa Ottawa, PhD. Thesis, 2014.
- [135] **Mola, O., Bauer, M.A.** (2011). Towards Cloud Management by Autonomic Manager Collaboration, Int. J. Communications, Network and System Sciences, 2011, 4, 790-802 doi:10.4236/ijcns.2011.432097, December 2011
- [136] **Kryftis, Y. Grammatikou, M., Kalogeras, D., Maglari, V.** (2017). Policy-Based Management for Federation of Virtualized Infrastructures, Journal of Network and Systems Management, April 2017, Volume 25, Issue 2, pp 229–252
- [137] **Waller, A., Sandy, I., Power, E., Aivaloglou, E., Skianis, C.** (2011). Policy Based Management for Security in Cloud Computing, FTRA International Conference on Secure and Trust Computing, STA 2011: Secure and Trust Computing, Data Management, and Applications pp 130-137.
- [138] Oracle Enterprise Management, Cloud Management, <https://cloud.oracle.com/management>, 2017
- [139] BMC Cloud Lifecycle Management, <https://docs.bmc.com/docs/cloudlifecyclemanagement>, 2017
- [140] **Outchakoucht A., Es-Samaali, H., & Leroy, J.P.** (2017). Dynamic Access Control Policy based on Blockchain and Machine Learning for the Internet of Things, (IJACSA) International Journal of Advanced Computer Science and Applications, 8(7), 417-424, <http://dx.doi.org/10.14569/IJACSA.2017.080757>
- [141] **Grabatin, M., Hommel, W., & Steinke, M.** (2018). Policy-Based Network and Security Management in Federated Service Infrastructures with Permissioned Blockchains, Security in Computing and Communications, 6th International Symposium, SSCC 2018, India, September 19–22, (pp. 145-156)
- [142] **Zhu X., Badr Y.** (2018). Identity Management Systems for the Internet of Things: A Survey Towards Blockchain Solutions, MDPI Sensors, 18(12): 4215, <https://doi.org/10.3390/s18124215>
- [143] **Maesa, D.F., Mori, P., and Ricci, L.** (2017). Blockchain Based Access Control, 17th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS), Jun 2017, Switzerland. Springer International Publishing, Lecture Notes in Computer Science, LNCS-10320, (pp. 206-220)
- [144] **Nuss, M., Puchta, A., Kunz, M.** (2018). Towards Blockchain-based Identity and Access Management for Internet of Things in Enterprises, 15th International Conference on Trust, Privacy and Security in Digital Business, TrustBus 2018, https://doi.org/10.1007/978-3-319-98385-1_12

- [145] **Ouaddah, A., Abou Elkalam A., and Ait Ouahman, A.** (2017). FairAccess: A new Blockchain-based access control framework for the Internet of Things, Security and Communication Networks, 9(18), 5943-5964, <https://doi.org/10.1002/sec.1748>
- [146] **Zhang, Y., Kasahara, S., Shen, Y., Jiang, X., Wan, J.** (2018) Smart Contract-Based Access Control for the Internet of Things, IEEE Internet of Things Journal, 6(2), 1594-1605, <https://doi.org/10.1109/JIOT.2018.2847705>
- [147] **Dorri, A., Kanhere, S.S., Jurdak, R., & Gauravaram, P.** (2017). Blokzincir for IOT security and privacy: The case study of a smart home, in 2017 IEEE International Conference on Pervasive Computing and Communications, Workshops (PerCom Workshops), March 2017, (pp. 618–623). <https://doi.org/10.1109/PERCOMW.2017.7917634>
- [148] **Zyskind, G., Nathan, O. & Pentland, A.** (2015). Decentralizing privacy: Using Blockchain to protect personal data, in 2015 IEEE Security and Privacy Workshops, May 2015, (pp. 180-184). <https://doi.ieeecomputersociety.org/10.1109/SPW.2015.27>
- [149] **Rantos, K., Drosatos, G., Demertzis, K., Ilioudis, C. & Papanikolaou, A.** (2018). Blockchain-based Consents Management for Personal Data Processing in the IoT Ecosystem, 15th International Joint Conference on e-Business and Telecommunications (ICETE 2018) – vol.2: SECRIPT, (pp. 572-577), <https://doi.org/10.5220/0006911007380743>
- [150] **Putz, B., Dietz, M., Empl, P., Pernulet, G.** (2020). EtherTwin: Blockchain-based Secure Digital Twin Information Management, Information Processing & Management, 58(1), <https://doi.org/10.1016/j.ipm.2020.102425>
- [151] **Esposito, C., Ficco, M., Gupta, B.B.** (2020). Blockchain-based authentication and authorization for smart city applications, Information Processing & Management, 58 (2), <https://doi.org/10.1016/j.ipm.2020.102468>
- [152] **Rouhani, S., Belchior, R., Cruz, R. S., and Deters, R.** (2020). Distributed Attribute-Based Access Control System Using a Permissioned Blockchain, arXiv preprint arXiv:2006.04384
- [153] **Banerjee, A., Joshi, K.P.** (2017). Link Before You Share: Managing Privacy Policies through Blockchain, 4th International Workshop on Privacy and Security of Big Data (PSBD 2017), USA, December 2017, (pp. 4438-4447), <https://doi.org/10.1109/BigData.2017.8258482>
- [154] OASIS: eXtensible Access Control Markup Language (XACML) version 3.0 (January 2013)
- [155] **Sinha PK.,** Distributed Operating Systems: Concepts and Design. IEEE Press: New York, NY, 1997.
- [156] **U.S. Department of Defense.** 1997b. DoD modeling and simulation (M&S) glossary, DoD 5000.59-M, December
- [157] **Page, E.H., Smith, R.,** (t.y) Introduction To Military Training Simulation: A Guide For Discrete Event Simulationists, Proceedings of the 1998 Winter Simulation Conference
- [158] **Page, E.H., Moose, R.L., Griffin, S.P.** (1997). Web-Based Simulation in SimJava using Remote Method Invocation, Winter Simulation Conference, Atlanta, GA, 7-10 December.22

- [159] **Jacobs, P.H.M, A. Lang, N., Verbraeck. A.** (2002). D-SOL; A Distributed Java Based Discrete Event Simulation Architecture, Proceedings of the Winter Simulation Conference, pages 793-800.14
- [160] **Oracle**, 2014. Microsystems, Inc., Java Management Extensions(JMX) Technology, available from <http://docs.oracle.com/technetwork/java/javase/tech/javamanagement-140525.html> June 2014.
- [161] **Barr, R., Haas, Z. J., Renesse, R.** (2005). JiST: An efficient approach to simulation using virtual machines, *Software Practice and Experience*, 35 2005, pp.539 – 576.
- [162] **Sulistio, A. Yeo, C.S, Rajkumar B.** (2004). A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools, *Software Practice And Experience*, 34:653–673 (DOI: 10.1002/spe.585)13
- [163] **Bagrodia, R., Meyer, R., Takai, M., Chen, Y., Song, H.** (1998). Parsec: A Parallel Simulation Environment for Complex Systems, Vol. 31(10), *IEEE Computer*, October 7.
- [164] **Howell F., McNab. R.** (1998). SimJava: A Discrete Event Simulation Package For Java With Applications In Computer Systems Modelling, *First International Conference on Web-based Modelling and Simulation*, San Diego, CA, Society for Computer Simulation, January.8
- [165] **Varga. A.** (2001). “The OMNeT++ Discrete Event Simulation System”, *European Simulation Multiconference (ESM 2001)*, June 6-9, Prague, Czech Republic.9
- [166] **Tyan H., Hou. C.** (2001). J-Sim JavaSim: A component based compositional network simulation environment, In *Western Simulation Multiconference*, January.10
- [167] **M.C. Little.** (2004). “JavaSim user guide”, public release 0.3. <http://javasim.ncl.ac.uk/>, June. 11
- [168] **Buyya R, Murshed M.** (2002). “GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing”, *Concurrency and Computation: Practice & Experience*, 14(13–15):1175–1220.4
- [169] **Legrand A., Marchal L., Casanova H.** (2003). Scheduling distributed applications: The SimGrid simulation framework, *Proceedings 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2003)*, Tokyo, Japan, 12–15 May.5
- [170] **Casanova, H., Legrand, A., Quinson, M.** (2008). SimGrid: a Generic Framework for Large-Scale Distributed Experiments, *10th IEEE International Conference on Computer Modeling and Simulation (UKSim)*, 2008.
- [171] **Song HJ, Liu X, Jakobsen D, Bhagwan R, Zhang X, Taura K, Chien A.** (2000). The MicroGrid: A scientific tool for modeling computational Grids, *IEEE Supercomputing (SC2000)*, Dallas, TX, 4–10 November.6
- [172] **Kilgore, R.A., Healy, K.J., Kleindorfer., G.B.** (1998). The future of Java-based simulation, In *Winter Simulation Conference*, pages 1707–1712, December.15
- [173] **Miller, J.A., Nair, S., Zhang, Z., Zhao. H.** (1997). JSIM: A JAVA-based simulation and animation environment, In *30th Annual Simulation Symposium (ANSS’97)*, Atlanta, USA.17

- [174] **Kreutzer, W., Hopkins, J., Mierlo, M.** (1997). SimJAVA - A Framework For Modeling Queueing Networks In Java, Proceedings of the Winter Simulation Conference 18
- [175] **Healy, K.J., Kilgore, R.A.** (1997). Silk: A Java-based process simulation language, In Winter Simulation Conference, 475–482, December.16
- [176] **Leon, E., Riesen, R., Maccabe, A., Bridges, P.** (2009). Instruction-Level Simulation of a Cluster at Scale, International Conference for High Performance Computing and Communications (SC), 2009.
- [177] **Bucy, J.S., Schindler, J., Schlosser, S.W., Ganger, G. R.** (2008). DiskSim Simulation Environment Version 4.0 Reference Manual, Tech. Rep. CMU-PDL-08-101, Carnegie Mellon University, Parallel Data Lab, 2008.
- [178] **Chen, W., Deelman, E.** (2012). WorkflowSim: A Toolkit for Simulating Scientific Workflows in Distributed Environments, 8th IEEE International Conference on eScience, 2012.
- [179] **Teng, F., Yu, L., Magoules, F.** (2011). SimMapReduce: A Simulator for Modeling MapReduce Framework, 5th FTRA International Conference on Multimedia and Ubiquitous Engineering (MUE), 2011, pp. 277 –282.
- [180] **Jung, J., Kim, H.** (2012). MR-CloudSim: Designing and Implementing MapReduce Computing Model on CloudSim, International Conference on ICT Convergence (ICTC), 2012, pp. 504 –509.
- [181] **Velho, P., Schnorr, L. M. , Casanova, H., Legrand, A.** (2013). On the Validity of Flow-level TCP Network Models for Grid and Cloud Simulations, ACM Transactions on Modeling and Computer Simulation 23 (4), 2013,
- [182] **Hoefler, T., Schneider, T., Lumsdaine, A.** (2010). LogGOPSim-Simulating Large-Scale Applications in the LogGOPS Model, ACM Workshop on Large-Scale System and Application Performance, 2010, pp. 597–604.
- [183] **Totoni, E., Bhatele, A., Bohm, E., Kale, L.** (2011). Simulation-based Performance Analysis and Tuning for a Two-level Directly Connected System, 17th Intl. Conference on Parallel and Distributed Systems, 2011, pp. 340–347.
- [184] **Malhotra, R., Jain, P.** (2013). Study and Comparison of Various Cloud Simulators Available in the Cloud Computing, Intl Journal of Advanced Research in Computer Science and Software Engineering, Vol 3, Issue 9, September 2013
- [185] **Nunez, A., Vazquez-Poletti, J., Caminero, A., Carretero, J., Llorente, I. M.** (2011). Design of a New Cloud Computing Simulation Platform, 11th International Conference on Computational Science and Applications, 2011, pp. 582–593.
- [186] **Bux, M., Leser, U.** (2013). DynamicCloudSim: Simulating Heterogeneity in Computational Clouds, DynamicCloudSim, SWEET '13, 2nd ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies, 2013, NY, USA
- [187] **Taufer, M., Kerstens, A., Estrada, T., Flores, D., Teller, P. J.** (2007). SimBA: A DES for Performance Prediction of Volunteer Computing Projects, 21st International Workshop on Principles of Advanced and Distributed Simulation, 2007, pp. 189–197.
- [188] **Estrada, T., Taufer, M., Reed, K., Anderson, D. P.** (2009). EmBOINC: An Emulator for Performance Analysis of BOINC Projects, Workshop on Large-Scale and Volatile Desktop Grids (PCGrid), 2009.
- [189] **Kondo, D.** (2007). SimBOINC: A Simulator for Desktop Grids and Volunteer Computing Systems, Available at <http://simboinc.gforge.inria.fr/>, 2007.

- [190] **Basu, A., Fleming, S., Stanier, J., Naicken, S., Wakeman, I., Gurbani, V. K.** (2013). The State of Peer-to-peer Network Simulators, *ACM Computing Survey*. 45 (4), 2013.
- [191] **Montresor, A., Jelasity, M.** (2009). PeerSim: A Scalable P2P Simulator, 9th International Conference on Peer-to-Peer, 2009, pp.99–100.
- [192] **Baumgart, I., Heep, B., Krause, S.** (2007). OverSim: A Flexible Overlay Network Simulation Framework, 10th IEEE Global Internet Symposium, 2007, pp. 79–84.
- [193] **Gil, T.M., Kaashoek, M.F., Li, J., Stribling, J.** (2005). P2PSim, a Simulator for Peer-to-Peer Protocols, Available at <http://pdos.csail.mit.edu/p2psim/>, 2005.
- [194] **Garcia, P., Pairot, C., Mondejar, R., Rallo, R.** (2004). PlanetSim: A New Overlay Network Simulation Framework, 4th Intl. Workshop on Software Engineering and Middleware (SEM), Vol. 3437 of LNCS, Springer, 2004, pp.123–136.
- [195] **Ranganathan, K.; Foster, I.** (2002). Decoupling computation and data scheduling in distributed data-intensive applications, 11th IEEE International Symposium on High Performance Distributed Computing, HPDC-11, 23-26 July, pp. 352–358.
- [196] **W. Depoorter, N. De Moor, K. Vanmechelen, J. Broeckhove,** Scalability of Grid Simulators: An Evaluation, 14th EuroPar Conference, Vol. 5168 of LNCS, Springer, 2008, pp. 544–553.
- [197] JFreeChart library, <http://www.jfree.org/jfreechart>
- [198] **Emeras, J., Ruiz, C., Vincent, J.M., Richard, O.** (2014). Analysis of the Jobs Resource Utilization on a Production System, *Job Scheduling Strategies for Parallel Processing Lecture Notes in Computer Science 2014*, pp 1-21.
- [199] **Ernemann, C., Song, B., Yahyapour, R.** (2003). Scaling of workload traces, *Job Scheduling Strategies for Parallel Processing. Volume 2862 of Lecture Notes in Computer Science*. Springer Berlin /Heidelberg, 2003, pp.166–182.
- [200] **Jacobs, P.H.M., Verbraeck, A.,** (2004). Single-threaded specification of process-interaction formalism in java, 36th Winter Simulation Conference. 2004, pp.1548–1555.
- [201] **Bowker, M.** (2014) IBM System x-Series M5:Infrastructure for the Cloud, Whitepaper, September 2014.
- [202] OpenStack documentation <http://docs.openstack.org/>
- [203] **Huang, K., Zeng, J. Huang, J.X.** (2014). IBM System x Solution for IBM Cloud Manager with OpenStack Reference architecture, IBM Systems and Technology Group, September 2014
- [204] **Watts, D., Bain, L., Leitenberger, C.** (2005). IBM Eserver xSeries and BladeCenter Server Management, March 2005.
- [205] IBM System x solution for IBM Cloud Manager with OpenStack Installation and configuration guide, IBM Systems and Technology Group ISV Enablement Team, September 2014.
- [206] IBM Cloud computing solutions: Transforming business with innovation ibm.com/systems/x/solutions/cloud/index.html
- [207] IBM Cloud Manager with OpenStack 4.1 Administrators Guide ibm.com/developerworks/wiki/smart-cloud-entry/documentation
- [208] ProActive, RMS System, <http://www.activeeon.com>

- [209] **Pelt, R., Jansen, S., Baars D. & Overbeek, S.** (2020). Defining Blokzincir Governance: A Framework for Analysis and Comparison, *Information Systems Management*, 38(1), 21-41, DOI: 10.1080/10580530.2020.1720046
- [210] **Rikken, O., Janssen, M., & Roosenboom-Kwee, Z.** (2019). Governance challenges of blockchain and decentralized autonomous organizations. *Information Polity*, 24(4), 397-417. <https://doi.org/10.3233/IP-190154>
- [211] **Scheid, E., Rodrigues B., & Stiller, B.** (2019). Toward a Policy-based Blockchain Agnostic Framework, *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, Arlington, VA, USA, pp. 609-613.
- [212] **Berdik, D., Otoum S., Schmidta, N., Portera, D., Jararweh, Y.** (2020). A Survey on Blockchain for Information Systems Management and Security, *Information Processing and Management*, 58 (1), 1-28, <https://doi.org/10.1016/j.ipm.2020.102397>
- [213] **Curran, B.** (2018). What is Blockchain Governance? Complete Beginner's Guide, <https://blockonomi.com/blokzincir-governance/>, September 21.
- [214] **Ehrsam F.** (2017), Blockchain Governance: Programming Our Future, <https://medium.com/@FEhrsam/blokzincir-governance-programming-our-future-c3bfe30f2d74>, November 2017, (Visited on 20 December 2020)
- [215] **Zamfir, V.** (2017). Serious concerns around on-chain governance moving forward, https://medium.com/@Vlad_Zamfir/against-on-chain-governance-a4ceacd040ca, December 2017, (Visited on 20 December 2020)
- [216] **Buterin, V.** (2017). Notes on Blockchain Governance, <https://vitalik.ca/general/2017/12/17/voting.html>, December 2017.
- [217] **Qureshi, H.** (2018). Blockchains should not be democracies, <https://hackernoon.com/blokzincirs-should-not-be-democracies-14379e0e23ad>, April 2018 (Visited on 20 December 2020)
- [218] **Wall, Larry D.** (2018). Blockchain Challenges and Governance, Notes from the Vault, (Visited on 20 December 2020) <https://www.frbatlanta.org/cenfis/publications/notesfromthevault/07-blokzincir-challenges-and-governance-2018-07-30>, July 2018
- [219] **Buterin, V.** (2017) The Meaning of Decentralization, <https://medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274>, February 2017, (Visited on 20 December 2020)
- [220] **Atzori, M.** (2017). Blockchain technology and decentralized governance: Is the state still necessary?, *Journal of Governance and Regulation*, 6(1), 45-62. http://doi.org/10.22495/jgr_v6_i1_p5
- [221] **Zachariadis, M., Hileman, G., Susan V.** (2019), Governance and control in distributed ledgers: Understanding the challenges facing Blockchain technology in financial services, *Information and Organization Journal*, 29, 105–117, <https://doi.org/10.1016/j.infoandorg.2019.03.001>
- [222] **Lumineau, F., Wang, W., Schilke, O.** (2020). Blockchain Governance-A New Way of Organizing Collaborations?, *Organization Science*, Oct 2020, 1-22, <https://doi.org/10.1287/orsc.2020.1379>
- [223] BIP, Bitcoin Improvement Proposals, <https://github.com/bitcoin/bips>
- [224] EIP, Ethereum Improvement Proposals, <https://eips.ethereum.org>

- [225] **Goodman, L.M.** (2014) Tezos- a self-amending crypto-ledger White paper, https://tezos.com/static/white_paper-2dc8c02267a8fb86bd67a108199441bf.pdf, September, 2014.
- [226] **Hanke, T., Movahedi, M. & Williams, D.** (2018). DFINITY Technology Overview Series Consensus System, <https://dfinity.org/static/dfinity-consensus-0325c35128c72b42df7dd30c22c41208.pdf>, January 2018.
- [227] Decred Community, Introduction to Decred Governance, <https://docs.decred.org/governance/introduction-to-decred-governance/> (Visited on 20 December 2020)
- [228] ISO/IEC 24760-1 – Information technology – Security techniques – A framework for identity management – Part1: Terminology and concepts’. ISO/IEC, 2011.
- [229] **Kalman, C., Toth, A.A.** (2019). Self-Sovereign Digital Identity: A Paradigm Shift for Identity, IEEE Security & Privacy, 17, 17-27, <https://doi.org/10.1109/MSEC.2018.2888782>
- [230] **Lesavre, L., Varin, P., Mell, P., Davidson, M., & Shook, J.** (2020). A Taxonomic Approach to Understanding Emerging Blockchain Identity Management Systems, <https://doi.org/10.6028/NIST.CSWP.01142020>
- [231] **Al-Bassam, M.** (2017). SCPKI: A Smart Contract-based PKI and Identity System. The ACM Workshop on Blockchain, Cryptocurrencies and Contracts, 2 April 2017; (pp. 35–40), <http://dx.doi.org/10.1145/3055518.305553>
- [232] **Liu, Y., Zhao, Z., Guo, G., Wang, X., Tan, Z., Wang, S.** (2017). An Identity Management System Based on Blockchain. 2017. Available online: <https://www.ucalgary.ca/pst2017/files/pst2017/paper-8.pdf>, (Visited on 20 December 2020).
- [233] **Axon, L.** (2015). Privacy-awareness in Blockchain-based PKI. Oxford University Research Archive, UK
- [234] **Augot, D., Chabanne, H., Clémot, O., George, W.** (2017). Transforming Face-to-Face Identity Proofing Into Anonymous Digital Identity Using the Bitcoin Blockchain. 2017. Available online: <https://arxiv.org/abs/1710.02951> (Visited on 20 December 2020).
- [235] **Augot, D., Chabanne, H., Chenevier, T., George, W., Lambert, L.** (2017). A User-Centric System for Verified Identities on the Bitcoin Blockchain. Data Privacy Management, Cryptocurrencies and Blokzincir Technology. DPM 2017, CBT 2017. Lecture Notes in Computer Science, vol 10436. (pp. 390-407), https://doi.org/10.1007/978-3-319-67816-0_22
- [236] **Brands, S. A.** (2000). Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy. MIT Press, Cambridge-London.
- [237] **Hardjono, T., Pentland, A.** (2019). Verifiable Anonymous Identities and Access Control in Permissioned Blockchains, <http://arxiv.org/abs/1903.04584> (Visited on 20 December 2020)
- [238] **Halpin, H.** (2017). NEXTLEAP: Decentralizing Identity with Privacy for Secure Messaging, 12th International Conference on Availability, Reliability and Security, ARES'17, 29 August–1 September 2017, (pp. 1-10). <https://doi.org/10.1145/3098954.3104056>
- [239] **Azouvi, S., Al-Bassam, M., Meiklejohn, S.** (2017). Who Am I? Secure identity registration on distributed ledgers, ESORICS/DPM/CBT-2017. LNCS, vol. 10436, (pp. 373–389), https://doi.org/10.1007/978-3-319-67816-0_21

- [240] UPort: Open Identity System for the Decentralized Web. Available online: <https://www.serto.id/> (Visited on 20 December 2020).
- [241] Shocard: Secure Enterprise Identity Authentication. Available online: <https://shocard.com/> (Visited on 20 December 2020).
- [242] Bitnation: Governance 2.0. Available online: <https://bitnation.co/> (Visited on 20 December 2020).
- [243] Civic Secure Identity Ecosystem—Decentralized Identity & Reusable KYC. Available online: <https://www.civic.com/> (Visited on 20 December 2020).
- [244] Decentralized Infrastructure for Self-Sovereign Identity. Available online: <http://jolocom.io/> (Visited on 20 December 2020).
- [245] Sovrin-Identity for All. Available online: <https://sovrin.org/> (Visited on 20 December 2020).
- [246] Self-Sovereign Identity. Available online: <https://www.evernym.com/> (Visited on 20 December 2020).
- [247] ID2020 Alliance, An Alliance Committed to Improving Lives through Digital Identity. Available online: <https://id2020.org/> (Visited on 20 December 2020).
- [248] **Vogelsteller, F.** (2017). ER725, Ethereum Identity Standard., October 2017, Available online: <https://github.com/Ethereum/EIPs/issues/725> (Visited on 20 December 2020).
- [249] W3C Decentralized Identifiers Specification. Available online: <https://w3c-ccg.github.io/did-spec/> (Visited on 20 December 2020).
- [250] **Reed, D., Sporny, M., Longley, D., Allen, C., Grant, R., Sabadello, M.** (2020). Decentralized Identifiers (DIDs) v1.0, Data Model and Syntaxes for Decentralized Identifiers (DIDs), Draft Community Group Report, <https://w3c-ccg.github.io/did-spec/>, 16 May 2020
- [251] **Sporny, M, Longley, D., Chadwick, D.** (2020). Verifiable Credentials Data Model 1.0, <https://w3c.github.io/vc-data-model/>, 15 January 2020
- [252] Decentralized PKI, Whitepaper, DPKI v1.0.0, December 2015, <https://github.com/WebOfTrustInfo/rwot1-sf/blob/master/final-documents/dpki.pdf>, (Visited on 20 December 2020)
- [253] **Zhang, B., Oliynykov, R., & Balogun, H.** (2019). *A Treasury System for Cryptocurrencies: Enabling Better Collaborative Intelligence*. Paper presented at The Network and Distributed System Security Symposium 2019, San Diego, United States. <https://doi.org/10.14722/ndss.2019.23024>
- [254] **Lalley, S.P., Glen Weyl, E.** (2018). Quadratic Voting: How Mechanism Design Can Radicalize Democracy, American Economic Association Papers and Proceedings, vol. 108, (pp. 33-37), DOI: 10.1257/pandp.20181002
- [255] **Buterin, V.** (2018). “Liberation Through Radical Decentralization”, May 2018, <https://medium.com/@VitalikButerin/liberation-through-radical-decentralization-22fc4bedc2ac>, (Visited on 20 December 2020).
- [256] **Dourlens, J.** (2017). Getting data from the internet with Oraclize, 28 September 2017, <https://ethereumdev.io/getting-data-internet-oraclize/>, (Visited on 20 December 2020)
- [257] POLICE, Blockchain Governance Model Project web portal, <https://github.com/bzlab/governance>, (Visited on 27 December 2020)

- [258] **Buterin V., et al.** (2014). “Ethereum white paper: a next generation smart contract & decentralized application platform”, First version, vol. 53, 2014. Retrieved from <https://github.com/ethereum/wiki/wiki/White-Paper>.
- [259] **Samman, G., Freuden, D.** (2020). “DAO: A Decentralized Governance Layer For The Internet Of Value”, May 2020
- [260] **Liu, L. Zhou, S., Huang, H., Zheng, Z.** (2020) From Technology to Society: An Overview of Blockchain-based DAO, arXiv:2011.14940v1 [cs.CR]
- [261] **Dupont Q.** (2017). “Experiments in Algorithmic Governance: A history and ethnography of The DAO, a failed Decentralized Autonomous Organization”, Bitcoin and Beyond, pp.157-177
- [262] **Valiente, M., Hassan, S., Pavón, J.** (2020). “Evaluating the Software Frameworks for Developing Decentralized Autonomous Organizations”, IEEE Access, 2020
- [263] ARAGON white paper, Available from: <https://github.com/aragon/whitepaper>
- [264] DAOStack white paper, Available from: <https://daostack.io/wp/DAOstack-White-Paper-en.pdf>
- [265] COLONY white paper, Available from: <https://colony.io/whitepaper.pdf>
- [266] **Donassolo, B., Casanova, H., Legrand, A., Velho, P.** (2010). Fast and Scalable Simulation of Volunteer Computing Systems Using SimGrid, 2nd Workshop on Large-Scale System and Application Performance (LSAP), 2010, pp. 605–612.
- [267] **Quinson, M., Rosa, C., Thiery, C.** (2012). Parallel Simulation of Peer-to-Peer Systems, 12th IEEE International Symposium on Cluster Computing and the Grid (CCGrid), 2012.
- [268] **Bedaride, P., Degomme, A., Genaud, S., Videau, B.** (2013). Toward Better Simulation of MPI Applications on Ethernet/TCP Networks, 4th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems, Denver, USA, 2013.
- [269] **DMTF,** (1999). Distributed Management Task Force, Specification for the Representation of CIM in XML, version 2.0, available from http://www.dmtf.org/download/spec/xmls/CIM_XML_Mapping20.php, 20 July 1999.



EKLER

EK A: POLICE Politika Dili grameri ve kaynak kodları

EK B: Blokzincir Yönetişim Akıllı Sözleşme Algoritmaları

EK C: Proposal Contract Oylama Fonksiyonu algoritması

EK D: PoC 2 İçin Örnek Politikalar





EK A: POLICE Politika Dili grameri ve kaynak kodları

Police dili grameri ve kaynak kodları <https://github.com/tdursun/police> adresinde kod deposu içerisinde dir.





EK B: Blokzincir Yönetişim Akıllı Sözleşme Algoritmaları

Çizelge B.1 : DIDRegistryContract ve Voting Contract sözde kod gösterilimi.

Registry Contract	Voting Contract
FounderDDOs # DDOs of founders. Created in Genesis Transactions. RO InitialDeveloperDDOs # DDOs of initial developers. Created in Genesis Transactions. RO (Read-only) DeveloperDDOs # DDOs of developers. DDOregistry # registered DDOs Reputations # reputations of experts function onboardActor (<i>DDO_of_Actor</i>) ▷ verify <i>DDO_of_Actor</i> ▷ add to registry DDOregistry.put(<i>DDO_of_Actor</i>) function onboardDeveloper (<i>DDO_of_Dev</i>) ▷ ensure that caller is a Founder ▷ verify <i>DDO_of_Dev</i> ▷ add to registry DeveloperDDO.put(<i>DDO_of_Dev</i>) function reportAbuse (<i>pid, did_of_reporter, did_of_misbehaved_actor</i>) ▷ verify claim ▷ freeze DID function getDDO(<i>did</i>) return DDOregistry[<i>did</i>] function getDeveloper(<i>did</i>) return DeveloperDDO[<i>did</i>] function isRegistered(<i>did</i>) return DDORegistry.has(<i>did</i>) function updateReputation(<i>did, value</i>) Reputations[<i>did</i>].add(<i>value</i>)	Proposals # all proposals proposed so far (Id, title, proposer, status, result, type, category, date, discussion link, etc.) Policies # active policy set registryContract #address of the RegistryContract # to propose a new policy function propose (<i>did_proposer, policy_to_add [, interval]</i>) ▷ verify DID of proposer ▷ lock stake of proposer ▷ create a new proposal contract (<i>proposal, this, regC</i>) ▷ announce “a new voting started” # start a new voting # to propose removal of a policy function propose (<i>did_proposer, policyid_to_remove</i>) ▷ lock stake of proposer ▷ create a new Proposal contract (<i>proposal</i>) ▷ start a new voting function listOfProposals () # retrieve list of proposal contracts function listActivePolicies () function withdrawProposal(<i>pid</i>) ▷ release locked stake ▷ seal proposal contract as read only function finalizeProposal (<i>proposal_id, policy, result</i>) ▷ do closing actions if <i>result</i> is YES then Policies ← update with <i>policy</i> ▷ trigger policy activation/deactivation end if Proposals[<i>proposal_id</i>].result ← <i>result</i> Proposals[<i>proposal_id</i>].completed ← <i>true</i> ▷ Unlock stake of proposer

Çizelge B.2 : Proposal Contact oylama fonksiyonu algoritması.

Proposal Contract
Proposal # proposal details (id, policy, discussion url, proposer, etc.). Encoded as a JSONObject Votes # accumulates votes submitted: (voter, vote, role, weight, accounts) Follows # follow orders of actors: (did_{follower}, did_{to_follow}, VCs, backup vote, accounts used) votingContract #address of the VotingContract registryContract #address of the RegistryContract construct (<i>proposal</i>, <i>vC</i>, <i>rC</i>) Proposal $\leftarrow$ <i>proposal</i>, votingContract $\leftarrow$ <i>vC</i>, registryContract $\leftarrow$ <i>rC</i>, function vote (<i>did_{voter}</i>, <i>vote</i>, <i>VCs</i>) # Explained in detail in Çizelge C.1 ▷ ensure that the voter owns a valid DID and she didn't vote before ▷ verify <i>VCs</i> against validity and binding ▷ verify delegation, if <i>VCs</i> include delegation credential(s) IF <i>VCs</i> includes <i>VC-Delegation</i> THEN ▷ ensure that the delegators didn't vote before with the stake(s) delegated ▷ verify delegation credential(s) and calculate $\sum \text{eligibleBalance}_{\text{delegator}}$ END IF ▷ store <i>vote</i> including eligible balance of both voter and delegator(s) Votes.push(<i>did_{voter}</i>, <i>role</i>, <i>vote</i>, ($\text{eligibleBalance}_{\text{voter}} + \sum \text{eligibleBalance}_{\text{delegator}}$), <i>VC.accounts</i>) ▷ If there is follower for <i>did_{voter}</i> process them for <i>follow</i> in Follows(<i>did_{voter}</i>) do weight $\leftarrow$ Calculate voting contribution of the follower Votes.push(<i>follow.did</i>, <i>follow.role</i>, <i>vote</i>, <i>weight</i>, <i>follow.VC.accounts</i>) end for function follow (<i>did</i>, <i>did2follow</i>, <i>VCs</i>, <i>backupVote</i>) ▷ ensure that <i>did</i> didn't cast a vote before ▷ confirm content of <i>VCs</i> and ensure that follower has possessions of accounts and verify role of the follower ▷ ensure that these accounts were not used in any other follow request weight $\leftarrow$ Calculate voting contribution of the follower IF <i>did2follow</i> already voted THEN Votes.push(<i>did</i>, <i>follower-role</i>, <i>vote-expert</i>, <i>weight</i>, <i>VC.accounts</i>) ▷ Update reputation value of the followed actor registryContract.updateReputation(<i>did2follow</i>, <i>weight</i>) ELSE ▷ store data to evaluate during wrap up actions of ballot Follows.push(<i>did</i>, <i>did2follow</i>, <i>follower_role</i>, <i>VCs</i>, <i>backupVote</i>, <i>VC.accounts</i>) function unfollow (<i>did</i>, <i>did2follow</i>, <i>didProof</i>) ▷ ensure that <i>didProof</i> is belongs to the caller ▷ ensure that <i>did2follow</i> didn't cast a vote yet ▷ cancel this follow request Follows.remove(<i>did</i>, <i>did2follow</i>) function closeVoting () ▷ check closing conditions (time, participation) ▷ take the postponed follow requests into account for <i>follow</i> in Follows do Votes.push(<i>follow.did</i>, <i>follow.role</i>, <i>follow.backupVote</i>, <i>balance</i>, <i>follow.VC.accounts</i>) end for ▷ count the votes (by weighting each vote with role, balance) in <i>Votes</i> and calculate decision of ballot ▷ trigger VotingContract and forward the result of ballot votingContract.finalizeProposal(<i>Proposal.id</i>, <i>result</i>, <i>Proposal.policy</i>) ▷ freeze/seal this contract

Not: Yönetişim Akıllı Sözleşmelerinin Solidity dilinde gerçeklemeleri ise <https://gist.github.com/tdursun/0fba066e6265a4d2c52a6bbac46030a6> adresinde verilmiştir.

EK C: Proposal Contract Oylama Fonksiyonu algoritması

Çizelge C.1 : Proposal Contact oylama fonksiyonu algortması.

Algorithm 1 Verify Vote

▷ *vote* denotes the decision of the owner of *did* on the proposal, *did* denotes the identity of the voter, *verifiableCredentials* denotes proof(s) the voter must fulfil and delegations, if any

function VOTE (*did*, *vote*, *verifiableCredentials*)

▷ ensure that the actor did not vote before for this proposal

assert (Votes[*did*] is not null)

▷ verify that the did belongs to a known actor

ddo ← null

ddo ← **registryContract**.getDDO(*did*)

assert (*ddo* is not null)

▷ verify that the *verifiableCredentials* are bound to this voting (against replay attacks)

assert (verifySign(encode(*vote*); *vote*.signature; *ddo*.publicKey))

assert (*vote*.vcProposalBinding = hash (*proposalID* || encode(*verifiableCredentials*)))

▷ process delegation credentials, if any, to determine delegated stakes to the voter

delegatedBalance ← 0, *accounts* ← []

for *vcDel* ← *verifiableCredentials*.getByType("Delegation") **do**

▷ authenticate the credential and ensure that the credential was issued to this voter

ddoDelegator ← **registryContract**.getDID(*vcDel*.issuer)

assert (verifySign(*vcDel*, *ddoDelegator*.publicKey))

assert (*vcDel*.delegate = *did*)

▷ ensure that the delegator didn't vote with this stake(s)

if Votes[*vcDel*.account] is null **then**

▷ accumulate delegated balance

delegatedBalance ← *delegatedBalance* + eligibleBalance(*vcDel*.account)

accounts.push(*pAddress*)

end if

end for

▷ verify that the credential(s) is complete, genuine and consistent with role

vcRole ← *verifiableCredentials*.getByType("ROLE")

role ← *vcRole*.getValue()

switch (*role*)

case "FOUNDER"

assert (**registryContract**.getFounder(*did*) is not null)

weight ← *COEFF*_{founder}

case "INITIAL_DEVELOPER"

assert (**registryContract**.getInitialDeveloper(*did*) is not null)

weight ← *COEFF*_{initial-dev}

case "DEVELOPER"

assert (**registryContract**.getDeveloper(*did*) is not null)

weight ← *COEFF*_{developer}

case "EXCHANGE_OWNER"

assert (verifySign (*vc*, *ddo*.publicKey))

vcA ← *verifiableCredentials*.getByType("AccountsPossession")

assert (verifySign (*vcA*, *ddo*.publicKey))

▷ ensure that EO has private keys for enough number of accounts with enough balance

sum, *accountsProcessed* ← calculateBalance (*vcA*.accountProofs, MIN_AMOUNT)

assert (*accountsProcessed*.size ≥ MIN_NUM_OF_EXCHANGE_ACCOUNT)

assert (*sum* ≥ MIN_EXCHANGE_AMOUNT)

▷ calculate voting weight with eligible balances of accounts specified in credential

weight ← calculateWeight (*vcA*.accountProofs, *delegatedBalance*, *accounts*, *COEFF*_{exchange})

Çizelge C.1 (devam): Proposal Contact oylama fonksiyonu algoritması.

Algorithm 1 Verify Vote

```

case "EXCHANGE USER"
     $vcA \leftarrow verifiableCredentials.getByType("AccountsPossession")$ 
    ▷ verify that vcA was issued by a valid Exchange Owner
     $sum, accountsProcessed \leftarrow calculateBalance(vcA.accountProofs, MIN\_AMOUNT)$ 
    assert ( $accountsProcessed.size \geq MIN\_NUM\_OF\_EXCHANGE\_ACCOUNT$ )
    assert ( $sum \geq MIN\_EXCHANGE\_AMOUNT$ )
    ▷ verify that Delegation and Role credentials were issued by a valid Exchange Owner
     $vcD \leftarrow verifiableCredentials.getByType("Delegation")$ 
     $ddoIssuer \leftarrow registryContract.getDDO(vcA:issuer)$ 
    assert ( $verifySign(vcD, ddoIssuer.publicKey)$ )
    assert ( $verifySign(vc, ddoIssuer.publicKey)$ )
    ▷ verify that the accounts were not used before for this proposal
     $sum, accountsProcessed \leftarrow calculateBalance(vcD.accountProofs, MIN\_AMOUNT)$ 
    assert ( $Vote[accountsProcessed]$  is null)
    ▷ calculate voting weight with eligible balances of accounts specified in credential
     $weight \leftarrow calculateWeight(vcD.accountProofs, delegatedBalance, accounts, COEFF_{exch-user})$ 
case "USER"
    assert ( $verifySign(vc, ddo.publicKey)$ )
     $vcA \leftarrow verifiableCredentials.getByType("AccountPossession")$ 
    assert ( $verifySign(vcA, ddo.publicKey)$ )
    ▷ calculate actor's weight with each account meeting eligible balance requirements
     $weight \leftarrow calculateWeight(vcA.accountProofs, delegatedBalance, accounts, COEFF_{user})$ 
case "MINER"
    assert ( $verifySign(vc, ddo.publicKey)$ )
     $vcA \leftarrow verifiableCredentials.getByType("AccountPossession")$ 
    assert ( $verifySign(vcA, ddo.publicKey)$ )
    ▷ verify that miner has at least one coinbase account with enough balance
     $hasMinerAccount \leftarrow false$ 
    for  $pAddress$  in  $vcA.accountProofs$  do
        if  $isCoinBaseAccount(pAddress)$  AND  $eligibleBalance(pAddress) \geq MIN\_AMOUNT$  then
             $hasMinerAccount \leftarrow true$ 
            break
        end if
    end for
    assert ( $hasMinerAccount$ )
    ▷ calculate voting weight with eligible balances of accounts specified in credential
     $weight \leftarrow calculateWeight(vcA.accountProofs, delegatedBalance, accounts, COEFF_{miner})$ 
end switch
    ▷ store the vote with its weight
     $Votes.push(did, role, vote, weight, accounts)$ 
    return true
end function
▷ calculates voting power of an actor by using eligible balance and role
private function  $calculateWeight(accountProofs, delBalance, accounts, userWeight)$ 
     $sum, accountsProcessed \leftarrow calculateBalance(accountProofs, MIN\_AMOUNT)$ 
     $accounts.merge(accountsProcessed)$ 
     $voteWeight \leftarrow (sum + delBalance) * userWeight$ 
    return  $voteWeight$ 
end function
▷ calculates  $\sum$  eligible balance by verifying both possession of accounts and eligible balance threshold
private function  $calculateBalance(accountProofs, threshold)$ 
     $sum \leftarrow 0, accs \leftarrow []$ 
    for ( $proof, pAddress$ ) in  $accountProofs$  do
        assert ( $verifySign(proof, pAddress)$ )
        assert ( $eligibleBalance(pAddress) \geq threshold$ )
         $sum \leftarrow sum + eligibleBalance(pAddress)$ 
         $accs.push(pAddress)$ 
    end for
    return  $sum, accs$ 
end function

```

EK D: PoC 2 İçin Örnek Politikalar

Çizelge D.1 : İkinci PoC 2 için hazırlanmış örnek politikalar (Makale içinde [12] yayınlanmış şekli ile verilmiştir).

Politikalar
Sample Policy1. The BIP-101 defining new maximum block size that grows over time at a predictable rate can be realized in our model by the following policy (The maximum size shall be 8.000.000 bytes at a timestamp of 2016-01-11 00:00:00 UTC, and shall double every 2 years, until 2036-01-06). <pre>inst mustdo /Policies/Consensus/policy1 { target t= /Blockchain; action t.setMaxBlockSize(calculateBlockSize(t.blockHeight)); when (Time.between("2016-01-11", "00:00:00", "2036-01-06", "00:00:00")); } calculateBlockSize(int block_timestamp){ time_start = 1452470400; time_double = 60*60*24*365*2; // 2 year size_start = 8000000; if (block_timestamp >= time_start + time_double*10){ //20 years later return (size_start * 2^10); }else{ time_delta = block_timestamp - time_start; return ((1+ (time_delta % time_double)/time_double)*size_start* 2^(time_delta/time_double)); //interpolate } }</pre>
Sample Policy 2. We specify another policy for BIP-102 which proposes one-time increase in total amount of transaction data permitted in a block from 1MB to 2MB. <pre>int BLCK_NUM = ...; inst mustdo /Policies/Consensus/policy2{ target t= /Blockchain; action t.setMaxTXDataSize(2*1024*1024); when (t.blockHeight = BLCK_NUM) ; } }</pre>
Sample Policy 3. EIP-170 (Hard fork) changing contract code size limit can be implemented as the following policy <pre>inst cando /Policies/SmartContract/policy3{ actor a=/SmartContracts; target t=/Blockchain; action t.createSmartContract(); //else contract creation fails with an out of gas error. when (t.blockHeight >= 2.675.000) AND (a.size <= 0x6000); // (2**14 + 2**13) } }</pre>
Sample Policy 4. EIP-160, EIP-1285, 1884, EIP-1109, EIP-150, EIP-1380 all are about defining new tariff for some opcodes. Thus, a policy proposal for EIP-150 could be as follows <pre>inst mustdo /Policies/VM/policy4{ target t=/VM; action t.setCommandPrice(EXTCODESIZE, 20), t.setCommandPrice(EXTCODECOPY, 20), t.setCommandPrice(BALANCE, 20), t.setCommandPrice(SLOAD, 50), t.setCommandPrice(CALL, 40), t.setCommandPrice(DELEGATECALL, 40), t.setCommandPrice(CALLCODE, 40), t.setCommandPrice(SELFDESTRUCT, 0); when (/Blockchain.blockHeight = 2,463,000) ; } }</pre>

Çizelge D.1 (devam) : İkinci PoC 2 için hazırlanmış örnek politikalar (Makale içinde [12] yayınlanmış şekli ile verilmiştir).

Politikalar

Sample Policy 5. The following policy is prepared for EIP-999 restoring the WalletLibrary contract with a new code:

```
inst mustdo /Policies/SmartContract/policy5{
    actor a=/SmartContracts;
    target t = /SmartContracts/SmartContract("0x863DF6BFa4469f3e...51c91A907b4");
    action t.setState(INIT),
    t.setCode("606060405234156200000...0b957e3e2942f861085beb9baab180029");
    when (/Blockchain.blockHeight = CNSTNTNPL_FORK_BLKNUM);
}
```

Sample Policy 6. The Difficulty Bomb (aka. the ice age), introduced in EIP #2 as in formula (7), is an artificial exponential increase in difficulty and makes it more difficult to keep the original chain alive after a hard-fork.

$$timeBomb = 2^{\lfloor blockNumber/100.000 \rfloor - 2} \quad (7)$$

Thus, EIP-1240, EIP-858, EIP-649, EIP-1227, EIP-1234, EIP-1276, and EIP-1295 propose reducing block rewards and delaying or removing difficulty bomb because of delays in planned upgrades or empowering users over blockchain's governance process again. For example, a policy for EIP-858, which delays difficulty bomb by 2.000.000 blocks as shown in formula (8) and adjusts block, uncle, and nephew rewards to reflect a new block reward of 1 ETH.

$$timeBomb' = 2^{\lfloor (blockNumber - delayBlockNumber)/100.000 \rfloor - 2} \quad (8)$$

```
inst mustdo /Policies/Blockchain/policy6{
    target t= /Blockchain;
    action t.setBlockReward(1), t.setNephewBlocReward(0),
    t.setDifficultyDelayParams(2.000.000);
    when (t.blockHeight = CNSTNTNPL_FORK_BLKNUM)
}
```

Sample Policy 7. A policy is given below for EIP-1108 reducing gas cost for elliptic curve precompiled smart contracts

```
int BLCK_NUM = ....;
inst mustdo /Policies/VM/policy7{
    target t=/VM;
    action t.setPrecompiledContractPrice(0x06, 150), //ECADD
    t.setPrecompiledContractPrice(0x07, 6000), //ECMUL
    t.setPrecompiledContractPrice(0x08, 34.000, 45.000); //Pairing check
    when (/Blockchain.blockHeight = BLCK_NUM);
}
```

Sample Policy 8. EIP-1010 transferring balances between accounts can be imitated with policy as follows:

```
int BLCK_NUM = ....;
inst mustdo /Policies/Accounts/policy8{
    Account a1=/Blockchain/Accounts/Account ("0xAb5801a7D398351b...259aeC9B");
    Account a2=/Blockchain/Accounts/Account ("0x15E55EF43efA834...4917e3c");
    target a1, a2;
    action a1.setBalance(a1.getBalance() -100.000); a2.setBalance(a2.getBalance() +100.000);
    //in ETH
    when (/Blockchain.blockHeight = BLCK_NUM);
}
```

ÖZGEÇMİŞ

Ad-Soyad

: Taner DURSUN

ÖĞRENİM DURUMU:

- **Lisans:** 1994, İstanbul Teknik Üniversitesi, Elektrik-Elektronik Mühendisliği Fakültesi, Kontrol ve Bilgisayar Mühendisliği Bölümü
- **Yüksek Lisans:** 1998, İstanbul Teknik Üniversitesi, Bilgisayar Mühendisliği ABD, Bilgisayar Mühendisliği Programı

MESLEKİ DENEYİM VE ÖDÜLLER:

- Öğretim Görevlisi, Atatürk Üniversitesi, 1995-1998
- Araştırmacı, TÜBİTAK UEKAE, 1998-2001
- Uzman Araştırmacı, TÜBİTAK UEKAE, 2001-2006
- Başuzman Araştırmacı, TÜBİTAK BİLGEM UEKAE, 2006-2022
- Kıdemli Başuzman Araştırmacı, TÜBİTAK BİLGEM UEKAE, 2022-...
- Misafir Öğretim Görevlisi, Hitit Üniversitesi, Sakarya Üniversitesi, 2023

DOKTORA TEZİNDEN TÜRETİLEN YAYINLAR, SUNUMLAR VE PATENTLER:

- **Dursun, T., Üstündağ, B.B.** 2021. A Novel Framework for Policy Based On-chain Governance of Blockchain Networks, *Information Processing and Management Journal*, vol 58, issue 4, <https://doi.org/10.1016/j.ipm.2021.102556>, July 2021.
- **Dursun, T., Dağ, H.** 2017. A Generic Framework for Building Heterogeneous Simulations of Parallel and Distributed Computing Systems, *Arab J Sci Eng*, Issue 8, 2017, DOI 10.1007/s13369-017-2497-z.
- **Dursun, T.** 2005. A Generic Policy Conflict Handling Model, *Lecture Notes in Computer Science*, LNCS vol 3733, 193-204, https://doi.org/10.1007/11569596_22
- **Dursun, T., Örencik, B.** 2003. POLICE: A Novel Policy Framework, *Lecture Notes in Computer Science*, 10.1007/978-3-540-39737-3_102 LNCS 2869 (2003) 819-827.
- **Dursun, T., Üstündağ, B.B.** 2021. A Conceptual Solution Proposal for Governance Challenges of DAOs, *International Conference on Computer Science, Machine Learning and Big Data*, December, 2021 India.
- **Dursun T., Dağ H.** 2014. A Generic Simulation Model For High Performance Computing Systems, *The International Conference on Modeling, Simulation and Visualization Methods (MSV14)*, Las Vegas, USA.

- **Dursun T.**, Dağ H. 2009. HeteroSim: Heterogeneous Simulation Framework, *The 12th Communications and Network Simulation Symposium (CNS09)*, San Diego, CA, USA.
- **Dursun, T.**, Örencik, B. 2004. POLICE Distributed Conflict Detection Architecture. *The IEEE International Conference on Communications (ICC04)*, Paris, June 20-26, Paris, France.
- **Dursun, T.**, Nasır, D., Örencik, B. 2004. Relation Based Policy Conflict Detection. *The 3rd Asia Pacific Int'l. Symp. on Information Technology (3rd APIS)*, İTÜ, Jan. 13-14, 192-198, İstanbul, Türkiye.
- **Dursun T.** 2004. POLICE PBM Framework, Poster Session in the Conference on the Human Impact and Application of Autonomic Computing Systems, (CHIACS2), March, IBM T. J. Watson Research Center, Yorktown Heights, New York, USA.
- **Dursun T.**, Örencik, B. 2003. Dağıtık Politika Çakışma Çözme Yaklaşımı, *1. Türkiye Ulusal Yazılım Sempozyumu (UYMS03)*, Ekim, Ege Üniversitesi, İzmir.
- **Dursun, T.**, Örencik, B. 2003. Distributed Policy Conflict Solution Approach, *The 20th Turkish National Informatics Congress*, pp.110-123, September 2-5, 2003, İstanbul.

DİĞER YAYINLAR, SUNUMLAR VE PATENTLER:

- **Dursun, T.** 2022. Metaverse Applications of Blockchain Technology, *Anahtar Journal*, Vol 397, pp.17, January 2022.
- **Dursun, T.** 2021. The Blockchain Technology, *İstihdamda3i Journal of Turkish Employment Agency*, Vol 35, November 2021.
- **Dursun, T.**, Birinci, F., Alptekin, B. 2021. Blockchain Technology for Supply Chain Management, *Global Joint Conference On Industrial Engineering and Its Application Areas GJCIE*, August 2020.
- **Dursun, T.**, Bingöl, M.A., Kalkar, Ö., Özcandan, N. 2020. Security and Privacy Issues of Blockchain, *Techical Report*, https://bkm.com.tr/wp-content/uploads/2015/06/blokszincirlerde_guvenlik_ve_mahremiyet.pdf, BKM - TÜBİTAK BİLGEM, April 2020.
- **Dursun, T.** 2020. Blokszincir Teknolojisi, *BİLGEM Teknoloji Dergisi*, Sayı 8, S.10-15, Ocak 2020.
- **Dursun, T.** Akdemir, Ö.K., Benzer, R. 2014. Robust Software Architecture for a Fiber Network Monitoring Sistem, *International Journal of Advanced Research in Computer Science and Software Engineering*, Vol.4, December 2014.
- Akdemir, Ö.K., **Dursun, T.**, Arslan, S. Akçayol, M.A. A GIS based novel active monitoring system for fiber networks, *Turkish Journal of Electrical Engineering & Computer Sciences*, DOI: 10.3906/elk-1306-217.
- **Dursun, T.**, Karabas, S., Örencik, B., Sogukpinar, İ. 2000. Secure Management of a Proxy Server using SNMP, *The Fifth Turkish Symposium on Computer Networks (BAS2000)*, pp.110-123, June 20-23, 2000, Bilkent University, Ankara.

- Gümüşkaya, H., Akbaş, E., **Dursun, T.**, Davulcu, O. 1999. Utilization of CORBA, Mobile Agents, Java, and Web Technologies for Network Management, *Regional Conference on Military Communication and Information Systems '99*, Vol.3 pp.25-34, October 6-8 , 1999, Zegrze, Poland.
- Gümüşkaya, H., Davulcu, O., Akbaş, E., **Dursun, T.** 1999. A CORBA and Java Mobile Agents Based Network Management Architecture, *International Symposium on Computer and Information Sciences XIV (ISCIS' 14)*, pp. 975-982, October 18-20, 1999, Kuşadası, İzmir.
- Gümüşkaya, H., **Dursun, T.**, Er, H. 1999. A Mobile Multi-Agent System Architecture for Intelligent Information Retrieval on the Web, 8. Turkish Symposium on Artificial Intelligence and Neural Networks (TAINN'99), June 23-25, 1999, Boğaziçi University, Istanbul.
- Gümüşkaya, H., **Dursun, T.** 1999. A Java Mobile Agents Based Network Management Architecture, *The Fourth Turkish Symposium on Computer Networks (BAS99)*, pp. 112-120, May 20-21, 1999, Istanbul Technical University, Istanbul.
- **Dursun, T.**, Örencik, B. 1998. BAĞ3 Group Communication Approach, *The Fifth Turkish Computer & Communication Symposium, BUTTİM*, Bursa, 18-22 November, 1998, s. 52-55.
- **Dursun, T.**, Örencik, B. 1998. Group Communication Methods for BAĞ3 Hierarchical Network Architecture, *The Third Turkish Symposium on Computer Networks.(BAS98)*, Bornova, İzmir, 25-26 June, 1998, ISBN: 975-441-126-3, s. 55-65