T.C.
EGE ÜNİVERSİTESİ
Fen Bilimleri Enstitüsü

ÇOKLU DEPO SİSTEMLERİNDE ÖLÇEKLENEBİLİR SORGU YÖNETİMİ

Burak YÖNYÜL

DOKTORA TEZİ

Danışman: Doç. Dr. Rıza Cenk ERDUR
İkinci Danışman: Prof. Dr. Oğuz DİKENELLİ

Bilgisayar Mühendisliği Anabilim Dalı
Bilgisayar Mühendisliği Doktora Programı

İzmir
2023

Burak YÖNYÜL tarafından doktora tezi olarak sunulan "Çoklu Depo Sistemlerinde Ölçeklenebilir Sorgu Yönetimi" başlıklı bu çalışma E.Ü. Lisansüstü Eğitim ve Öğretim Yönetmeliği ile E.Ü. Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi'nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 14.09.2023 tarihinde yapılan tez savunma sınavında aday oybirliği/oyçokluğu ile başarılı bulunmuştur.

Jüri üyeleri **İmza**

Jüri Başkanı : Doç. Dr. Rıza Cenk ERDUR

Raportör Üye : Dr. Öğr. Üyesi Özgür GÜMÜŞ

Üye : Doç. Dr. Övünç ÖZTÜRK

Üye : Doç. Dr. Özgün YILMAZ

Üye : Doç. Dr. Korhan KARABULUT

Üye : Prof. Dr. Oğuz DİKENELLİ

EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

ETİK KURALLARA UYGUNLUK BEYANI

EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliğinin ilgili hükümleri uyarınca Doktora Tezi olarak sunduğum "Çoklu Depo Sistemlerinde Ölçeklenebilir Sorgu Yönetimi" başlıklı bu tezin kendi çalışmam olduğunu, sunduğum tüm sonuç, doküman, bilgi ve belgeleri bizzat ve bu tez çalışması kapsamında elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara atıf yaptığımı ve bunları kaynaklar listesinde usulüne uygun olarak verdiğimi, tez çalışması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını, bu tezin herhangi bir bölümünü bu üniversite veya diğer bir üniversitede başka bir tez çalışması içinde sunmadığımı, bu tezin planlanmasından yazımına kadar bütün safhalarda bilimsel etik kurallarına uygun olarak davrandığımı ve aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul edeceğimi beyan ederim.

14/09/2023

İmzası

Burak YÖNYÜL

ÖZET

ÇOKLU DEPO SİSTEMLERİNDE ÖLÇEKLENEBİLİR SORGU YÖNETİMİ

YÖNYÜL, Burak

Doktora Tezi, Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı: Doç. Dr. Rıza Cenk ERDUR

Tez İkinci Danışmanı: Prof. Dr. Oğuz DİKENELLİ 14.09.2023, 183 sayfa

Büyük veri; sahip olduğu hız, çeşitlilik, büyüklük ve doğruluk boyutlarıyla yönetilmesi güç bir olgudur. Büyük veriyi yönetebilmek için tüm bu boyutların belirli ölçülerde ele alınarak çözülmesi gereklidir. Sunulacak olan çözümde büyük boyuttaki veri, yapısına uygun model(ler)deki depo veya depolarda saklanmalı, hızlı sorgulanmanlı, ayrıca doğru ve güvenilir kaynaklardan elde edilmiş olmalıdır. Tam da bu noktada çoklu depo sistemleri büyük verinin yönetimini sağlamak için çözüm olarak ortaya çıkmaktadır. Sistem temel olarak bir çoklu depo ortamı ve bu ortamı depo özelinde tekil veya tümleşik olarak sorgulayan bir sorgu motorundan oluşur. Buradan hareketle büyük veri yönetimine bir çözüm olarak tez kapsamında bir alan odaklı çoklu depo sistemi oluşturulması hedeflenmiştir. Öncelikle yönetilecek büyük veri için sağlık alanı belirlenerek veri kümesi olarak farklı yapılarda ve büyük boyutta veri içeren MIMIC-III seçilmiştir. MIMIC-III'e referans veren tüm çalışmalar taranmış ve önemli kullanım durumu senaryoları belirlenmiştir. Sistemin değerlendirilmesi için; bu senaryolardan tekil ve birleşik sorgular oluşturulmuştur. Böylece verinin doğruluğu ve yapısallığından emin olunduktan sonra içinde bulunduğu alan, büyüklüğü ve yapısı özelinde çözümlenerek bölüneceği veri modelleri belirlenmiştir. Bu veri modellerine uygun saklama teknolojileri seçilmiş ve tüm ilişkili MIMIC-III verisi bu depolara yüklenmiştir. Kurulan bu mimarinin etkinliğini ortaya koymak adına; işleticiler ve dönüştürücüler ile birlikte aktör model yapısı üzerinde ölçeklenebilir bir çoklu depo sorgu motoru geliştirilmiştir. Sistem sorgu işletim süreleri ve

ölçeklenebilirlik bakış açılarından değerlendirilmiştir. Sonuçlar çoklu depo sistemlerinin belirli bir alandaki büyük veriyi etkin biçimde yönetebildiğini kanıtlamakta ve hem geliştirilen sistemin hem de bu araştırma alanının önemini ortaya koymaktadır.

Anahtar sözcükler: Çoklu Depo Sistemleri, Sorgu İşleme, Sorgu Çalıştırma, Sorgu Motorları, Ölçeklenebilirlik, Büyük Veri Bütünleştirme, Büyük Veri Yönetimi.

ABSTRACT

SCALABLE QUERY MANAGEMENT IN POLYSTORE SYSTEMS

YÖNYÜL, Burak

Ph.D. in Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Rıza Cenk ERDUR

Second Supervisor: Prof. Dr. Oğuz DİKENELLİ 14.09.2023, 183 pages

Big data is a difficult phenomenon to manage by having velocity, variety, volume and veracity dimensions. In order to manage big data, it is necessary to take into account all these dimensions in a certain manner. In the solution to be presented, large-scale data should be persisted in the store(s) suitable for its structure, it should be queried quickly, and it should also be obtained from accurate and reliable sources. At this point, multistore / polystore systems emerge as a solution to manage the big data. The system basically consists of a polystore environment and a query engine that queries this environment as simple or federated. From this point of view, it is aimed to create a domain orineted polystore system within the scope of the thesis in order to provide an exemplary big data management. First of all, the medical domain has been determined for the big data to be managed, and MIMIC-III has been chosen as the dataset which contains large amount of data with various structures. All studies that reference MIMIC-III have been scanned and critical use case scenarios have been determined. For the evaluation of the system; single and compound queries have been generated from these scenarios. Thus, after proving the correctness and structure of the data, its models have been analyzed and determined according to the field, size and structure. Storage technologies suitable for these data models have been selected and all MIMIC-III data associated with these stores have been imported. In order to demonstrate the effectiveness of the proposed architecture, a scalable polystore

query engine along with the executors and wrappers for the determined stores has been built on top of the actor model. The system has been evalauted in terms of the query evaluation times and scalability point of views. The results prove that the polystore systems are able to manage the big data in a relevant domain effectively, and put forward both the importance of the developed system and this research field.

Keywords: Polystore Systems, Query Processing, Query Execution, Query Engines, Scalability, Big Data Integration, Big Data Management.

ÖNSÖZ

Bugüne kadar süregelen bilim yolculuğumun tamamında lisans öğrenimimden bu yana danışmanım olan, kıymetli fikirleri ve desteği ile bana yön veren saygıdeğer hocam Doç. Dr. Rıza Cenk ERDUR'un vermiş olduğu büyük emeklere ve rehberliğine minnetarım. Akademik dünyaya adım attığımdan bu zamana ikinci danışmanım ve çok kıymetli hocam Prof. Dr. Oğuz DİKENELLİ'nin vizyonu, öngörüleri, fikrî önderliği ve açtığı yenilikçi ufuklar ile Seagent Laboratuvarı'nda çalışma arkadaşlarım ile birlikte çok sayıda araştırma çalışması yürüttük. Bu süreçte bireysel çalışmadan öte bir takım oyuncusu olmanın ve sinerji oluşturarak daha büyük işler başarabilmenin kıymetini kavradık.

Doktora sürecinde almış olduğum Büyük Veri dersi ile ortaya konulan doktora tezinin temel fikirleri atılmıştır. Burada saygıdeğer hocam Prof. Dr. Murat Osman ÜNALIR'ın vizyonu ve dersi alan arkadaşlarımızın katkılarıyla yaptığımız konferans yayını çalışmasının önemli etkisi olmuştur. Lisans tezinden bu yana süregelen; uygulamalı bilimlere, veri yönetimine ve sorgulamaya olan merakım, ders aldığım değerli hocalarımın katkıları ve doktora süresince ölçeklenebilir sorgulama üzerine sürdürülen araştırmalar ve gönderilen dergi yayını çalışması sonucunda tez kapsamında geliştirilen çoklu depo sistemi ortaya çıkmıştır.

İZMİR
14/09/2023

Burak YÖNYÜL

İÇİNDEKİLER

Sayfa

İÇ KAPAK ii
KABUL VE ONAY SAYFASI iii
ETİK KURALLARA UYGUNLUK BEYANI v
ÖZET vii
ABSTRACT ix
ÖNSÖZ xi
İÇİNDEKİLER xiii
ŞEKİLLER DİZİNİ xvii
TABLOLAR DİZİNİ xix
SİMGELER VE KISALTMALAR DİZİNİ xxi
1 GİRİŞ 1
1.1 Motivasyon ve Tezin Önemi 1
1.2 Amaç ve Özgün Değer 2
1.3 Tezin Akışı ve İçeriği 4
2 KAVRAMSAL LİTERATÜR VE MİMARİ TEMELLERİ 5
2.1 Veri Bütünleştirme 5
2.2 Veri Webini Sorgulama ve Bağlı Veri Bütünleştirme 7
2.3 Çok Çeşitli Kalıcılık (Polyglot Persistence) 9
2.4 Aktör Model 10
2.4.1 Aktör Tabanlı Küme Parçalama Yöntemi 11
2.5 Birleşim (Join) Yöntemleri 17
3 İLİŞKİLİ ÇALIŞMALAR 22
3.1 Çoklu Depo Sistemleri (Multistore/Polystore Systems) 22
3.1.1 CloudMDSQL 23
3.1.2 BigDAWG 28
3.1.3 Myria 32

3.1.4 Apache Drill 32
3.1.5 ESTOCADA 33
3.1.6 FORWARD 34
3.1.7 Diğer Özgün Yaklaşımlar 36
4 YÖNTEM 45
4.1 Veri Kümesi Seçimi ve Analizi 45
4.1.1 MIMIC-III Veri Kümesinin Literatürdeki Önemi 45
4.1.2 MIMIC-III Veri Modeli Analizi 46
4.1.3 MIMIC-III Veri Kümesi Kullanım Çözümlemesi 53
4.1.4 Verinin İlişkisel Depoya Yüklenmesi 54
4.1.5 MIMIC-III Kullanım Senaryoları 55
4.2 Çoklu Depo Ortamının Oluşturulması 60
4.2.1 Veri Depolarının Seçilmesi 61
4.2.2 MIMIC-III Verisinin Seçilen Depolara Yüklenmesi 65
4.3 Ölçeklenebilir Sorgu İşleme ve Birleşim Mekanizması 78
4.3.1 Sorgu İşleme Altyapısının Belirlenmesi 79
4.3.2 Birleşim Algoritmasının Belirlenmesi 80
4.3.3 Aktör Tabanlı SPARQL Sorgulama Mimarisi 85
4.3.4 AQuA'ya Sorgu Takibi Özelliğinin Kazandırılması 89
4.3.5 MonARCh: Sparql Sorgu Sonucu Değişikliği İzleme Mimarisi . . . 92
4.3.6 Sorgu İşletim Motorunun Çoklu Depo Sistemine Bütünleştiril-mesi 98
5 DEĞERLENDİRME 103
5.1 Çoklu Depo Sistemlerinin Özellik Bakımından Kıyas-lanması . . . 104
5.2 Sınama Sorgu Kümesinin Oluşturulması 106
5.2.1 Önemli Kullanım Senaryolarının Seçimi 107
5.2.2 Kullanım Senaryolarından Sorgu Çıkarımı 111
5.3 Sorgu Sonuçları ve Çözümleme 138
5.3.1 Tekil Sorgu Sonuçları 139
5.3.2 Çoklu Depo Birleşik Sorgu Sonuçları 148
5.4 Ölçeklenebilirlik Çözümlemesi 152
5.5 Sistemin Kısıtları 155

6 SONUÇ VE GELECEK ÇALIŞMALAR 159
KAYNAKLAR DİZİNİ 163
TEŞEKKÜR 181
ÖZGEÇMİŞ 183

ŞEKİLLER DİZİNİ

Şekil **Sayfa**

2.1 Üç Düğümlü Örnek Bir Küme Parçalama Mimarisi 13
2.2 Bir Bilgisayar Düğümü Üzerinde Yer Alan Aktörler ve Parça Temsili 13
2.3 Varlık Aktörü Topluluğunun Küme Parçalama ile Küme Üzerinde Temsili 14
2.4 Küme Parçalama Sisteminde Bir Düğümün Çökmesi Durumu . . . 15
2.5 Küme Parçalama Sistemi Üzerinde Mesaj İşletimi-1 16
2.6 Küme Parçalama Sistemi Üzerinde Mesaj İşletimi-2 16
2.7 Küme Parçalama Sistemi Üzerinde Mesaj İşletimi-3 17
2.8 İç İçe Döngülü Birleşim (Nested Loop Join) Algoritması 17
2.9 Bağlamalı Birleşim (Bind Join) Algoritması 18
2.10 Sırala-Birleştir Birleşim (Sort-Merge Join) Algoritması 19
2.11 Karma Birleşim (Hash Join) Algoritması 20
3.1 CloudMDSQL Sorgu Motoru Mimarisi 25
3.2 BigDAWG Çoklu Depo Sistemi Mimarisi (Gadepally et al., 2016) . 30
3.3 BigDAWG Ara Katmanı (Gadepally et al., 2016) 30
3.4 ESTOCADA Mimarisi (Bugiotti et al., 2015a) 34
3.5 FORWARD Mimarisi 35
4.1 Paralel GRACE Karma Birleşim (Hash Join) Algoritması 83
4.2 AQuA Mimarisi ve Sorgu İşleme Akışı 87
4.3 MonARCh'ın Sorgu İzleme Akışı 94
4.4 MonARCh için Örnek Küme Yapısı Modeli 96
4.5 AQuAPooL Mimarisi ve Sorgu İşleme Akışı 100
4.6 AQuAPooL için Örnek Küme Yapısı Modeli 102
5.1 Postgresql vs AQuAPooL Zaman Serisi Tekil Sorgu Sonucu Kıyaslaması (s cinsinden) 141

5.2 Postgresql vs AQuAPooL Anahtar Değer Tekil Sorgu Sonucu Kıyaslaması (ms) 144
5.3 Postgresql vs AQuAPooL Arama-Belge Tekil Sorgu Sonucu Kıyaslaması (s) 148
5.4 Postgresql vs AQuAPooL Çoklu Depo Birleşik Sorgu Sonucu Kıyaslaması (s) 152
5.5 Farklı sorgu seçiciliklerine göre ağ üzerinden iletilen sorgu başına toplam mesaj boyutu 153
5.6 Farklı Tipteki Sorgu Seçiciliklerine göre MonARCh'ın Sorgu İzleme Kapasitesi 154

TABLOLAR DİZİNİ

Çizelge **Sayfa**

4.1 AQuA aracının FedBench Sorgularının İşletimi üzerinden WoDQA ile kıyaslanması 88
5.1 AQuAPooL ve Güncel Çoklu Depo Sistemlerinin Son Teknoloji Özellikler ve Gereksinimler Bakımından Kıyaslanması 106
5.2 Postgresql ve AQuAPooL Zaman Serisi Tekil Sorgu Sonuçları (s cinsinden) 142
5.3 Postgresql ve AQuAPooL Anahtar Değer Tekil Sorgu Sonuçları (ms) 145
5.4 Postgresql ve AQuAPooL Arama-Belge Tekil Sorgu Sonuçları (s) . . 147
5.5 Postgresql ve AQuAPooL Çoklu Depo Birleşik Sorgu Sonuçları (s) . 151

SİMGELER VE KISALTMALAR DİZİNİ

Simgeler	Açıklama
LAV	Local as View
GAV	Global as View
$AQuAPooL$	Actor based Querying Architecture for Polystores on the Linked data
$AQuA$	Actor based Querying Architecture
$MonARCh$	Monitoring Architecture for sparql Result Changes
$MIMIC$	Medical Information Mart for Intensive Care
CPT	Current Procedural Terminology
ICU	Intensive Care Unit
DRG	Diagnosis Related Group
ICD	International Classification of Diseases
ECG	Electrocardiogram
EMR	Electronic Medical Record
RDF	Resource Description Framework
GCS	Glasgow Coma Scale

1 GİRİŞ

1.1 Motivasyon ve Tezin Önemi

Veri; modeli, yapısı, ait olduğu domain, büyüklüğü, sorgulanması vb. birçok boyutu itibariyle başlı başına yönetilmesi güç bir bileşendir. Dolayısıyla bu kadar çok boyutlu bir olgunun yönetimini genel geçer yargılara dayandırmak yanlış olacaktır. Öncelikle içinde yer aldığı domain göz önünde bulundurularak kendisini oluşturan ve geliştiren sistem içerisinde çözümlenmelidir. Buna göre tek bir saklama teknolojisinin kullanıldığı bir senaryo veri çeşitliliğin olmadığı bir domain içerisinde düzgün çalışabilecek iken, veri çeşitliliğinin yoğun olduğu ve boyut olarak sürekli genişleyen bir başka alanda çok yetersiz kalıp çökecektir. Günümüz bilgi sistemlerinde işlenen veri hacminin her geçen gün artmasıyla ölçeklenebilirlik gereksinimi doğmuş ve bu gereksinime göre çok büyük miktarlardaki veriyi performans kaybı olmadan saklayıp sorgulayabilmek önem kazanmıştır. Bu ihtiyaçtan ötürü her biri farklı kullanım amacına sahip NoSQL veri modelleri ortaya çıkmıştır. "Herkese uyan tek bir beden yok" ("No one size fits all") ilkesinden yola çıkılarak farklı yapıdaki verilere ihtiyaç duyulan bir sistemde tek bir veri modeli yaklaşımının kullanılması uygun olmayacaktır. Çok çeşitli kalıcılık (polyglot persistence) yaklaşımına göre bir veri yönetim sisteminde farklı gereksinimleri karşılayan farklı veri modellerinin bir arada ve bütünleşik biçimde yönetilmesi gerekmektedir (Sadalage and Fowler, 2013). Bu gereksinim de veri bütünleştirme problemini doğurmaktadır. Farklı ve özel şemalardaki verinin bütünleşik yönetilip sorgulanması amacıyla çoklu depo sistemleri (polystore systems) ortaya çıkmıştır. Bu sistemlerde belirli bir domaindeki verinin farklı veri yapılarındaki alt parçalara bölünerek yönetilmesini sağlamak amaçlanmaktadır.

Çoklu depo sistemleri büyük verinin 3 temel karakteristiği olan çeşitlilik, hız ve boyut özelliklerinin yönetiminin kolaylaştırılmasını hedefler. Bu özelliklerin içinden hedef alınan ana karakteristik çeşitliliktir. Bilgi sistemlerindeki farklı gereksinimlerden ötürü, sistemin bütününü oluşturan veriler parça parça

farklı yapılarda saklanabilmektedir. Bilginin bir bütün olarak sorgulanabilmesi için de bu parçaların birbiriyle ilişkilendirilerek bütünleştirilmesi gerekmektedir. Büyük verideki çeşitlilik ihtiyacının ortaya çıkmasındaki nedenlerin başında da veriye hızlı erişim gelmektedir. Buna göre sistemdeki bilginin bütününü farklı veri modellerindeki parçalara ayırırken, bu parçalarda yer alacak olan alt bilginin parçadaki ilgili veri modeline uyumluluğunun yanı sıra, alt bilgiye erişimin de bu veri modeli ile hızlı olarak sağlandığından emin olunmalıdır. Burada çeşitlilik ve hız birbiriyle uyumlu şekilde işlemelidir ki veriye en hızlı erişimi sağlayan veri modeli her zaman en uygun olacak diye bir kural yoktur. Dolayısıyla bir veri modeline uygun saklama çözümünün hızlı olmasının yanında, kendisinde saklanan verinin de bu modele uygun olması verinin doğasına uygun şekilde kolaylıkla sorgulanmasına da olanak tanımaktadır. Son olarak büyük verinin adıyla ilişkili olarak verinin boyut olarak fazla olması durumunda da veriye hızlı ve ölçeklenebilir şekilde erişilebilmesi esastır. Dolayısıyla büyük boyuttaki veriyi yönetmesi için seçilecek bir veri saklama çözümünün buna uygun bir altyapıya ve yeteneğe sahip olması esastır.

1.2 Amaç ve Özgün Değer

Bir veri modellemesi ve dağıtımı (distribution) yapılırken önceklikle bu verinin yer aldığı alan ve bu alana özgü gereksinimler çözümlenmelidir. Bu gereksinimlere çözüm sunmak için ortaya çıkan çoklu depo sistemlerindeki temel bileşenlerden ilki çoklu depo ortamıdır. Eldeki verinin boyutu, içerdiği alana özgü ilişkiler ve çözümlenen gereksinimler ışığında karmaşıklığı düşük seviyede tutmak için mümkün olan en az sayıda veri modeli ve bu modele özgü veri depoları belirlenip kurulmalıdır. Böylece çoklu depo sisteminin diğer bileşeni olan sorgu motoru, oluşturulan bu veri ortamı üzerinde etkin şekilde çalışabilecektir. Bir diğer yandan veriye erişim hızı yani verinin sorgulama performansı sistemin olmazsa olmaz gereksinimidir. Bunu sağlamak için de çoklu depo sistemlerinde yer alan sorgu motorunun kendisine gelen basit ve birleşik sorguları hızlı şekilde işleterek çabuk sonuç üretebilmesi gerekmektedir. Buna göre hem sorguları dağıtık ve ölçeklenebilir çalıştıran, hem de sorgu

sonuçlarını etkin ve performanslı şekilde birleştirebilen bir sorgu işleme altyapısının oluşturulması önem arz etmektedir.

Bu tez kapsamında odaklanılacak başlıca nokta belirli bir alana ve bu alandaki veri gereksinimlerine özgü bir çoklu depo sisteminin tasarlanması, gerçekleştirimi ve etkinliğinin kanıtlanmasıdır. Bu bağlamda sağlık alanı seçilmiş ve veri kümesi ise medikal yoğun bakım verilerini içeren ve birçok çalışma tarafından bu alanda kullanılan başlıca referanslardan olan MIMIC-III olarak belirlenmiştir. MIMIC-III verisinin gereksinim çözümlemesi yapılarak üstün yöntemler (best practices) ile veri modelleri ve her bir modele özgü veri depoları belirlenerek bir çoklu depo ortamı oluşturulmuştur. Yine üstün yöntemler kullanılarak MIMIC-III verisinin bu çoklu depo ortamına (data distribution) dağıtımı gerçekleştirilmiştir.

Tez kapsamında odaklanılan bir diğer ana nokta ise çoklu depo sistemine gelecek sorguların etkin ve ölçeklenebilir şekilde yönetilerek çalıştırılması olmuş, sistemdeki veri ekleme ve güncelleme işlemlerinin yönetimi ise tez kapsamına dahil edilmemiştir. Sorgu işletimi bakış açısından sunulan özgün katkılardan biri çoklu depo ortamında etkin şekilde ve paralel olarak çalışabilen bir hash birleşim (join) algoritmasının gerçekleştirimidir. Buna göre tez çalışmasına paralel bir araştırma çalışması olarak yürütülüp gerçekleştirilen aktör tabanlı sorgu takip ortamı geliştirilip iyileştirilerek ölçeklenebilirlik yeteneği kazandırılmıştır. Tasarlanan çoklu depo sistemine gelecek sorguların yüksek performansta işletilmesi ve ayrıca sistemin eşzamanlı ve yoğun şekilde gelen sorguları ölçeklenebilir ve yüksek verimlilikte (high throughput) yönetmesi hedeflenmiştir. Dolayısıyla çoklu depo sisteminin temelindeki sorgu motoru da, bahsi geçen sorgu takip ortamından ayrıştırılarak oluşturulduğundan performans olarak aynı yeteneklere ve özelliklere sahiptir.

Geliştirilen sistemin değerlendirilmesi (evaluation) için MIMIC-III veri kümesine referans veren önemli çok sayıda çalışma taranarak gerçekçi medikal senaryolar çıkarılmış, derlenmiş ve sınıflandırılmıştır. Kritik öneme sahip ve çok çalışılan senaryolar arasından tekil ve birleşik sorgulardan oluşan bir sorgu kümesi oluşturulmuş ve çoklu depo sisteminin performansı bu sorgular üze-

rinden sınanmıştır. Sistemin ölçeklenebilirlik yeteneği, miras alınan (inherited from) sorgu takip ortamı çalışmasında ortaya konmuş olup onunla aynı sınırlara sahiptir. Dolayısıyla, çoklu depo sisteminin sorgu işletim performansı ve ölçeklenebilirlik sonuçları, böyle bir mimari yapılanmanın seçilen alandaki veri tüketimi gereksinimi için kritik öneme sahip ve elzem olduğunu göstermiştir.

1.3 Tezin Akışı ve İçeriği

Tezin devamında Bölüm-2'de tez çalışmasının temellerini oluşturan kavramsal literatür açıklanmaktadır. Bölüm-3'te ilişkili çalışmalar literatürde üzerinde yoğun olarak çalışılan önemli çoklu depo sistemleri gruplandırılarak incelenmiştir. Bölüm-4'te tezin gerçekleştirilmesinde kullanılan yöntemler, sistem mimarisi tasarımı, gerçekleştirim aşamaları ve işleyişi anlatılmaktadır. Bölüm-5'te ise geliştirilen sistemin değerlendirilmesi adına yapılan çalışmalar anlatılmakta olup, çoklu depo sisteminin sorgu işletimi sonuçları, ölçeklenebilirliği ve kısıtları ortaya konmakta, açıklanarak tartışılmaktadır. Bölüm-6'da ise tez kapsamında ortaya konan alana özgü ve ölçeklenebilir bir çoklu depo sistemi mimarisinin, elde edilen sonuçlar ile literatürdeki gerekliliği ve önemi ortaya konmakta ve temelinde yatan fikirlerin somut delillere dayandığı açıklanarak sonuçlandırılmaktadır. Ayrıca gelecekte sisteme eklenmesi planlanan işlevsellikler ve üretilmesi planlanan yayın çalışmalarına da değinilmektedir.

2 KAVRAMSAL LİTERATÜR VE MİMARİ TEMELLERİ

Bu bölümde tez çalışmasının mimari temellerini oluşturuna kavramsal literatür özetlenmektedir.

2.1 Veri Bütünleştirme

Veri Bütünleştirme farklı kaynaklardaki verinin bir araya getirilerek, bu verinin kullanıcıya birleştirilmiş bir biçimde sunulması problemine odaklanan alandır. İlişkisel veri tabanlarının ortaya çıkışından beri çalışılan bir konu olmakla birlikte, veritabanı sistemleri varlığını sürdürdükçe de çalışılmaya devam edecektir. Bu yaklaşımı teorik yönleriyle inceleyen (Lenzerini, 2002) temel olarak; bir veri bütünleştirme uygulamasının modellenmesi, veri bütünleştirmede sorgu çalıştırma, tutarsız veri kaynaklarıyla başa çıkma ve sorgular üzerinde çıkarsama konuları üzerinde durmuştur. Veri bütünleştirme sistemleri global şema (global schema) ve kaynaklar kümesi (set of sources) olmak üzere iki temel bileşeni baz alan bir mimari üzerine kurulmaktadır. Yazılım mimarisi açısından Local As View (LAV) ve Global As View (GAV) olmak üzere iki ana yaklaşım bulunmaktadır. Çalışmada bu iki yaklaşım modelleme, sorgu işletimi, tutarsız kaynakların ele alınması ve sorgu çıkarsama açılarından değerlendirilmiş ve birbiri ile kıyaslanmıştır.

LAV yaklaşımında global şema, kaynaklardan bağımsız olarak tanımlanır ve global şema ile kaynaklar arasındaki ilişkiler ise her bir kaynağı global şemanın görüntüsü (view) olarak tanımlayarak kurulur. LAV yaklaşımı global şemanın sabit ve iyi tanımlanmış olduğu durumlarda (bir kurumsal model veya ontoloji olarak) avantajlıdır. Global şema değişmediğinden yeni bir kaynak sisteme kolayca eklenebilir ve sistem kolayca genişletilebilir. Yeni kaynağın diğer kaynaklar ile bağlantısı global şema üzerinde tanımlanır. Sorgu işletimi view'lar üzerinden eşlemelerin tahmin edilmesi ile yapılır ve tam (exact) sonuç garanti edilmez. Verinin bütünleştirilmesi global şemadaki eşlemelere göre

sorgu işlemcisi tarafından ele alınır.

GAV yaklaşımında ise global şema, kaynaklar cinsinden ifade edilerek oluşturulur. GAV yaklaşımı kaynakların sabit olduğu ve sorgu çalıştırımlarının yoğun olduğu veri bütünleştirme sisteminde etkilidir. Sorgu işletimi LAV yaklaşımına göre daha basittir. Global şema ile yerel şemalar arasındaki sorgu ve veri dönüşümü sarmalayıcılar (wrappers) aracılığıyla gerçekleştirilir. Her alt sorgu ilişkili olduğu kaynak üzerinde işletilip sonuçları sarmalayıcılar tarafından dönüştürüldükten sonra birleştirilerek ana sorgu sonucu elde edilir. Sorgu, sarmalayıcılar tarafından ele alındığından dönüşüm önceden bellidir ve tam sonuç garanti edilir. Ancak sisteme yeni kaynak eklemek zordur, çünkü yeni eklenen şema için veriyi ve sorguyu dönüştürecek bir sarmalayıcı yazılması gereklidir.

Halevy et al. (2006) yaptıkları çalışma kapsamında veri bütünleştirme çalışmalarını derlemişlerdir. Burada veri bütünleştirme kavramı; şema eşlemelerinin yaratılması, uyarlanabilir sorgu işleme, XML'in kullanımı, model yönetimi, uçtan uca veri yönetimi ve yapay zekâ yönleriyle değerlendirilmiş ve bu alanlarda yapılan çalışmalara değinilmiştir. Ayrıca veri bütünleştirme endüstrisinin çalışma alanları ve karşılaşabilecekleri ölçeklenebilirlik, performans, kapsam ve yapay zekâ araçlarının kullanımı gibi zorluklar tartışılmıştır. Çalışmada ayrıca LAV yaklaşımı ile yeni bir kaynağın ekleme kolaylığı ve kaynak şemalar üzerinde kısıtların kolaylıkla tanımlanabilmesi avantajları vurgulanmıştır. Değinilen bir diğer önemli nokta gelecek çalışmalar kısmında bahsedilen veri uzaylarının (dataspace) "ihtiyacın kadar kullan" ilkesine dayanan veri yönetim şeklidir. Buna göre şema eşlemeleri kapsamlı olarak düşünülmeden her bilgi kaynağı birer servis olarak sunulmakta ve sistem kullanılmaya başlandıktan sonra ihtiyaca göre şema eşlemeleri eklenebilmektedir.

Tez kapsamında gerçekleştirilen sistemde birbirine bağlı birden çok veri modeli yer almaktadır. Dolayısıyla bu modelleri, modeller arası ilişkileri ve modeller üzerindeki sorgu işletimini yönetebilmek için bir veri bütünleştirme yaklaşımının tasarlanması gerekmektedir. Tasarlanan sistemde veri bütünlüğü önemli bir yere sahiptir ve her veri kaynağının veri yapısı sistem gerçekleştirilme-

den önce belirlidir. Ayrıca sistemin yoğun bir şekilde sorgulanması öngörülmektedir. Bu açıdan GAV yaklaşımının kullanılmasının daha uygun olacağı oldukça açıktır.

2.2 Veri Webini Sorgulama ve Bağlı Veri Bütünleştirme

Bağlı Veri kavramı, ilk olarak Tim Berners-Lee tarafından yayınlamış olduğu teknik bildiride (Berners-Lee, 2006) ortaya atılmıştır. Bu bildiride, Web'de yayınlanan verilerin, genellikle "Verinin Ağı" (Web of Data) olarak adlandırılan küresel bilgi alanının bir parçası haline getirilmesinin temel ilkeleri ortaya konmuştur (Bizer et al., 2011). O zamandan beri, farklı alanlardan kuruluşlar verilerini Bağlı Veri ilkelerini[1] izleyerek yayınlamaya başladıkça Verinin Ağı kademeli olarak genişlemiştir. Günümüzde Verinin Ağı, yönetim, medya, eğlence, yaşam bilimleri ve coğrafi yerler gibi belirli alanları kapsayan çok sayıda birbiriyle ilişkili veri kümesi içermektedir. Bunlara ek olarak DBpedia[2] (Auer et al., 2007; Bizer et al., 2009; Lehmann et al., 2015) gibi birden çok alana yayılan veri kümeleri de bulunmaktadır. Bu veri kümelerinin tümü, Bağlı Açık Veri (Linked Open Data - LOD) Bulutu [3] olarak adlandırılır.

Verinin Ağı büyüdükçe zaman içerisinde sorgulama gereksinimi doğmuştur ve bu gereksinim de SPARQL'in ortaya çıkmasını sağlamıştır. Bugün Bağlı Veri için ortak kabul gören sorgulama dili olan SPARQL, 15 Ocak 2008'de resmi bir W3C[4] Tavsiyesi haline gelmiştir (Prud'hommeaux et al., 2017). SPARQL 1.0, bireysel veri kümelerinin SPARQL uç noktaları aracılığıyla sorgulanmasına izin vermektedir. Fakat, önceleri birkaç SPARQL uç noktasından gelen verileri aynı anda birleştirebilecek birleştirilmiş sorgular çalıştırmanın hiçbir yolu yoktu. Bugün, birleştirilmiş (federated) sorguları işleme yeteneği, Web üzerinden dağıtılan Bağlı Veri'yi tüketmek için açık bir gereklilik haline gelmiştir. SPARQL'e sürüm 1.1'de birleştirilmiş sorgulama yeteneği eklenmiştir (Arenas

[1] https://www.w3.org/DesignIssues/LinkedData
[2] https://www.dbpedia.org/
[3] https://lod-cloud.net
[4] https://www.w3.org

and Pérez, 2012).

Bölüm-3'te anlatılacak çoklu depo sistemi literatüründe (Bonaque et al., 2016; Karimov et al., 2019; Khan et al., 2019; Poudel et al., 2019, 2022) gibi birçok önemli ve güncel çalışma veri bütünleştirme için RDF şemasından faydalanmaktadırlar. Dolayısıyla Bölüm-2.1'de bahsedilen veri bütünleştirme yöntemleri, belirlenen GAV (Global as View) yaklaşımı ve Bağlı Veri'nin yapısal ve yapısal olmayan veriyi bütünleştirici yeteneği göz önünde bulundurulduğunda tez kapsamında da kullanılacak global şema olarak RDF[5] seçilmiştir. Buna göre sistemde yer alacak yerel şemalar üzerindeki verinin global şema olan RDF'e dönüşümlerini gerçekleştiren sarmalayıcılar yazılmıştır. Tezde katkı olarak sorgu işletimine odaklanıldığından sarmalayıcılar yalnızca sonuç (result) dönüşümlerini gerçekleştirmekte olup, sorgu dönüşümü özelliği bulunmamaktadır. Bu açıdan ana sorguda doğrudan yerel depoların kendi sorguları kullanılmış olup birleşim (join) işlemi bu sorgular üzerinden ortak tanımlanmış değişkenler ile yapılmaktadır.

Bunun dışında doktora öğrenimi boyunca tez konusuna paralel olarak; SPARQL sorgularının sonuç değişimlerini izleyen bir mimarinin geliştirilmesi çalışması yürütülmüştür. Bu mimarinin temelinde birleştirilmiş SPARQL sorguları alt sorgulara ayrılarak sonuç değişimleri, ilgili RDF veri kümesinden uçnokta aracılığı ile belirli zaman aralıklarında sorgulanmaktadır. Bağlı veri ortamı, çoklu depo veri ortamı ile benzerlik göstermekte olup tek bir veri modeline sahip olması kısmında ayrışmaktadır. Sorgu takip ortamı mimarisinin temelinde ise paralel ve dağıtık çalışan, ayrıca SPARQL bağımsız çalışabilecek şekilde dönüştürülebilecek yapıda bir sorgu işletim ardışık düzeni (pipeline) bulunmaktadır. Böylece tez kapsamında geliştirilen çoklu depo sistemi için her deponun veri modeline özgü çalıştırıcılar (executors) ve dönüştürücüler (transformators), sorgu takip ortamından ayrıştırılan sorgu işletim ardışık düzenine eklenmiş ve sistemin sorgu işleme altyapısı tamamlanmıştır.

[5]https://www.w3.org/RDF

2.3 Çok Çeşitli Kalıcılık (Polyglot Persistence)

Büyük veri, ilişkisel veri modeli yaklaşımıyla ölçeklenemeyen verinin verimli bir şekilde depolanmasına, yönetilmesine ve işlenmesine olanak tanır (Manyika et al., 2011). Büyük verinin hız (velocity), hacim (volume) ve çeşitlilik (variety) olmak üzere 3 temel karakteristik özelliği bulunmaktadır (Berman, 2013). Büyük verinin bu özelliklerinden hız; birim zamanda kaynaklar arasında aktarılan veri miktarını, hacim; çok büyük miktarlardaki veriyi, çeşitlilik; farklı yapılara sahip verileri ifade etmektedir. Buna göre farklı tiplerde verilerin bulunduğu, bu verinin zaman içinde sürekli arttığı ve artan veri miktarından bağımsız hızlı sorgulama ve saklama ihtiyacı duyulan bir bilgi sistemi büyük verinin bu üç temel karakteristik özelliğiyle doğrudan bağlantılı olacaktır. "Herkese uyan tek bir beden yok" ("No one size fits all") sözü birçok farklı soruna çözüm getirebilecek tek bir yöntem olamadığı durumları belirtmek için kullanılan genel geçer bir ifadedir. Bu ifadenin veritabanı literatüründe "çok çeşitli kalıcılık" ilkesi (polyglot persistence) ile doğrudan ilişkili olduğu söylenebilir. Buna göre farklı saklama ihtiyaçlarını karşılayan tek bir saklama teknolojisi de yoktur. İşte bu nedenle, farklı veri saklama ihtiyaçlarını yönetebilmek için farklı saklama teknolojileri kullanılmalıdır (Sadalage and Fowler, 2013). Veritabanı sistemleri, temelinde bağlı bulundukları şemaya göre; ilişkisel (relational), anahtar-değer (key-value), sütun (columnar), belge (document), çizge (graph), zaman serisi (time-series) vb. olarak özelleşip sınıflanmaktadır. Farklı tipte şemaya sahip depoların (store) birlikte ve bütünleşik olarak yönetilmesi veritabanı literatüründe çoklu depo (multistore veya polystore) olarak ifade edilmektedir.

Bu bakış açısıyla doktora öğrenimi sırasında Büyük Veri dersi kapsamında geliştirilen çalışmada (Ünalir et al., 2015) büyük veri içeren bilgi sistemlerinin yönetimi için çok modelli bir mimari yaklaşım ortaya atılmıştır. Çok çeşitli kalıcılık üzerine kurgulanan ve LAV yaklaşımına dayanan mimarinin temelinde bir çoklu depo sistemi önerisi olduğu söylenebilecektir. Buna göre farklı modeller arası veri ve sorgu dönüşümleri, bir ontoloji olarak tanımlanan

global şemada yer alan eşlemeler üzerinden, anahtar değer ortak veri modeli aracılığıyla gerçekleştirilmektedir. Ortaya konan bu çalışma çoklu depo sistemleri üzerine kurgulanan tez çalışmasının fikirsel temellerinin atılmasında önemli rol oynamıştır.

2.4 Aktör Model

Farklı veritabanlarındaki veriyi etkin şekilde sorgulamak ve sorgu sonuçlarını birleşim (join) amacıyla ağ üzerinden iletmek için, mimarinin omurgası olarak güçlü bir mesajlaşma mekanizmasının oluşturulması gerekmektedir. Dolayısıyla, sorgu motoru mimarisinin de bu paralelde ölçeklenebilir, dağıtık ve eşzamanlı olması gerekir. Programlama açısından, iş parçacığı tabanlı bir yaklaşım, bu gereksinim için bir çözüm olarak mükemmel bir şekilde uyacaktır. Bu ifadeler ve gerçekler ışığında aktör model (Hewitt et al., 1973; Hewitt, 2010) eşzamanlı hesaplamayı (concurrent computation) ele alan nihai çözüm olarak ortaya çıkmaktadır. Aktör, posta kutusu aracılığıyla mesajları kabul eden ve aldıkları mesajın türüne göre tepkisel olarak bazı hesaplamalar yapan ilkel hesaplama birimidir. Aktörler ayrıca mesajlar göndererek birbirleriyle işbirliği yapacak şekilde tasarlanmıştır. Aktör model hata toleransını, her bir hata koşulundan kaçınmaktan ziyade, hatanın nasıl düzeltileceğini bilen danışmanına (supervisor) bırakmayı öneren "bırakın çöksün" (let it crash) felsefesiyle desteklemektedir. Bu ve benzeri diğer birçok faydalı özelliği ile aktör model, iş parçacığı programlamanın daha güvenli ve model tabanlı bir gerçekleştirimidir. Buradan hareketle sistemin aktör model üzerine kurulmasına karar verilmiştir. Uygulama dili olarak Scala kullanılmış ve JVM üzerinde aktör model gerçekleştirimi olarak Akka araç takımı (toolkit)[6] seçilmiştir. Akka'nın küme parçalama özelliği sayesinde sistem yüksek oranda ölçeklenebilir ve eşzamanlı hale gelmiştir. Böylece çoklu depo sorgu motoru yüksek performansa sahip olup, ayrıca binlerce CPU-bağımlı aktöre ve bilgisayar kümesi düğümü (cluster node) başına on binlerce msg/sn'ye (msg/s) kadar

[6]https://akka.io

çıkabilen mesaj yükü patlamasına karşı dirençlidir.

2.4.1 Aktör Tabanlı Küme Parçalama Yöntemi

Ölçeklenebilirlik kavramı iki boyutta ele alınmaktadır. Dikey ölçeklenebilirlik donanımsal boyutta olup, fiziksel bir bilgisayarın işlemci, RAM, disk vb. kaynaklarının artırılması demektir. Tez kapsamında ele alınan ölçeklenebilirlik ise yatay ölçeklenebilirlik olup, bir uygulama, servis veya işletim sisteminin çeşitli stratejilere dayanarak birden fazla bilgisayarın kaynaklarından faydalanabilmesi yeteneğidir. Sorgu motoru kapsamında uygulama bazında ölçeklenebilirlik yeteneği sağlanmıştır.

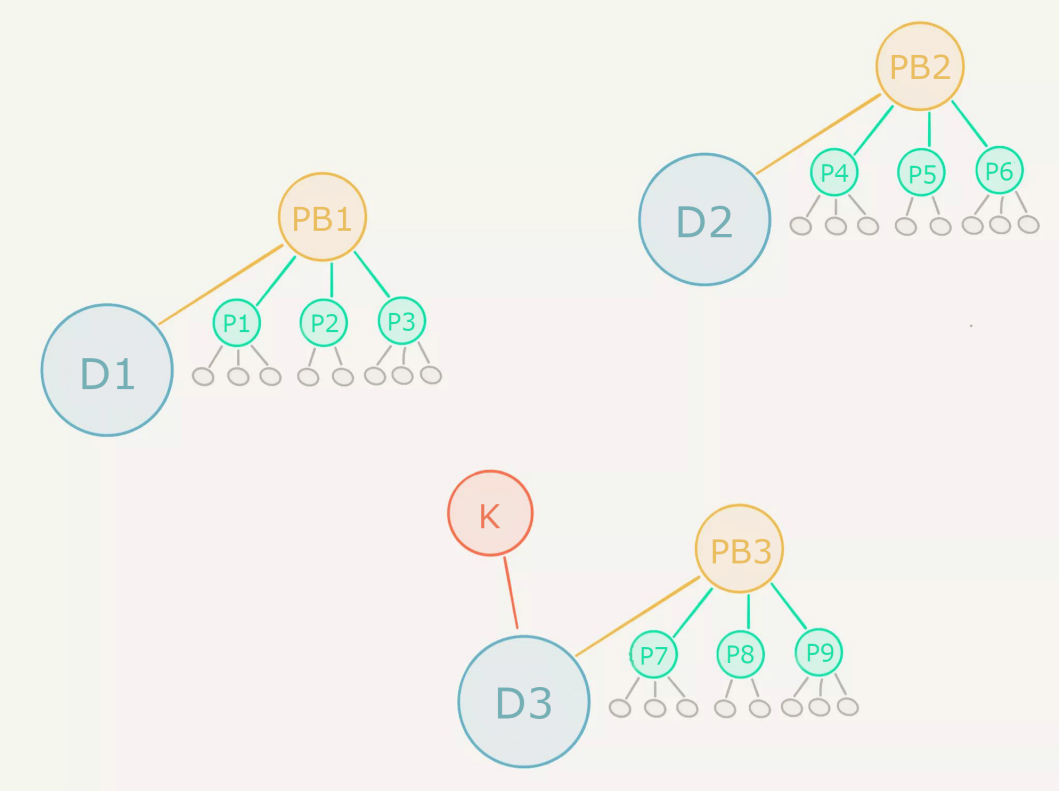
Sorgu motorunun omurgası aktör mimarisi üzerine inşa edilmiş ve gerçekleştirim altyapısı olarak da Akka kütüphanesi seçilmiştir. Akka araç takımı (toolkit) aynı anda işletim (concurrency), yönlendirme (routing), kendi kendini iyileştirme (self-healing), kümeleme ve parçalama ile ölçeklenebilirlik (clustering & sharding) gibi birçok özelliği bünyesinde barındırmaktadır. Bünyesinde Akka kütüphanesinin küme parçalama (cluster sharding) özelliği kullanılan sorgu motoru doğası gereği birden çok bilgisayar üzerinde dağıtık çalışabilecek yapıya kavuşmuştur. İlerleyen paragraflarda Akka'nın küme parçalama özelliği detaylı olarak incelenip açıklanmaktadır.

Küme parçalama; aktörlerin kümeyi oluşturan düğümler (bilgisayarlar) üzerinde otomatik olarak dağıtılmasını sağlar. Düğümler üzerinde dağıtılan aktörler ile, zaman içerisinde değişebilecek fiziksel lokasyonlarının bilinmesine gerek kalmadan, global tekil tanımlayıcıları (unique identifier) aracılığıyla iletişime geçilir. Akka küme parçalamanın temel fikirleri (Helland, 2016)'ın çalışmasından gelmektedir. Buradaki kavramsallıktan faydalanılarak aktörlere "varlık (entity)" adı verilmiştir. Eğer uygulama içinde, birlikte tek makineye sığamayacak kadar kaynak kullanan aktörler bulunuyorsa küme parçalama yöntemi oldukça elverişli olacaktır. Bu kapsamdaki parçalama kavramı, id'ye sahip aktörlerin, yani varlıkların otomatik olarak kümedeki birçok düğüm (node=bilgisayar) üzerinde dağıtılması işlemidir. Her varlık aktörü yalnızca tek bir düğüm üzerinde çalışır ve bu aktöre mesaj yollayanların aktörün

fiziksel yerini bilmeleri gerekmez. Bu özellik, varlık tanımlayıcısının (entity id) yer aldığı mesajı varış noktasına nasıl yönlendireceğini bilen parça bölgesi aktörü (Shard Region actor) ile sağlanır. Varlık aktörleri kalıcı (persistent) tipte tanımlanarak hata durumunda veya varlık aktörlerinin fiziksel yerlerinin değişmesi gerektiğinde, var olan durumlarını saklayarak (storing) yeniden başlatıldıklarında eski durumlarını geri kazanmaları sağlanabilmektedir.

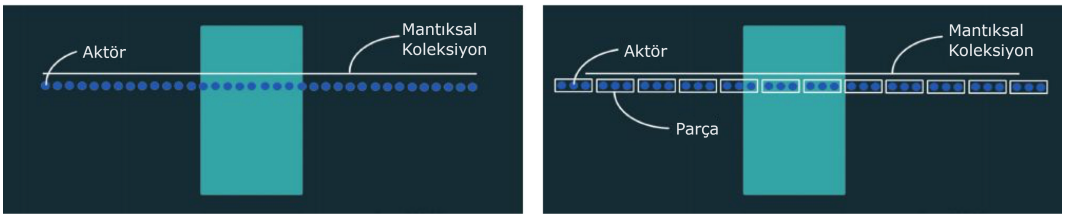
Küme parçalama kullanılırken başlangıçta her varlık tipi kümedeki her düğüm için kaydettirilir. Böylece parça bölgesi aktörünün referansı elde edilerek mesajların bu aktör aracılığıyla iletilmesi sağlanır. İlgili varlıklara erişebilmek için gerekli olan ve parça tanımlayıcısının ve varlık tanımlayıcısının bulunmasını sağlayan extractShardId ve extractEntityId metotları her varlık tipi için ayrı ayrı tanımlanır. Bir parça (shard) birlikte yönetilen varlıklar grubudur. Gruplama extractShardId fonksiyonu ile yapılır. Belirli bir varlık tanımlayıcısı için parça tanımlayıcısı hep aynı olmalıdır, aksi durumda bir aktör aynı anda birden fazla yerde çalışmaya başlayabilir.

Küme parçalama mimarisinin ölçek-farkında tarafının arkasında; parçalar, parça bölgeleri ve bir koordinatör yatmaktadır. Parçalar basit ifadeyle varlıklar grubudur, bir arada yer alırlar ve böylece daha sonra taşınmaları da kolaylaşır. Her düğümde yalnızca bir tane parça bölgesi aktörü yer alır. Bu aktörler parçalanmış aktörlerin alanına geçiş için bir yerel temsilci görevi görürler. Son olarak da tüm kümede yalnızca bir tane bulunan, küme tekil (cluster singleton) modda çalışan ve küme hakkında global kararlar veren parça koordinatörü bulunmaktadır. Örnek olarak hangi parçanın hangi bölgede yer alıp çalışacağını belirleyen parça koordinatör aktörüdür. Şekil-2.1'de örnek bir küme parçalama mimarisi basit olarak temsil edilmektedir. Burada temsil edilen kısaltmaların açıklamalarında **K**: Koorindatör, **D**: Düğüm, **PB**: Parça Bölgesi, **P**: Parça anlamına gelmektedir.



Şekil 2.1: Üç Düğümlü Örnek Bir Küme Parçalama Mimarisi

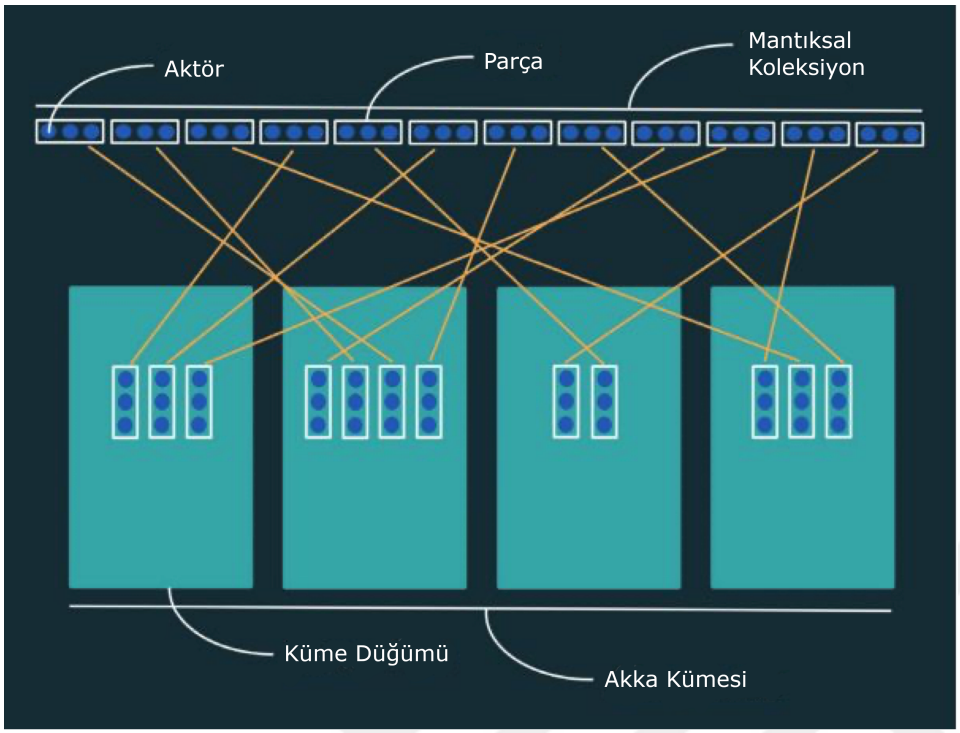
Aşağıda küme parçalama özelliği ve işleyiş mekanizması şekiller üzerinden anlatılmaktadır. Şekil-2.2'de bir düğüm ve bu düğüm üzerinde yer alan varlık aktörleri temsil edilmiştir. Şekilden görüldüğü üzere tüm bu varlıklar tek bir düğümde yer almak için oldukça fazla sayıdadırlar. Ayrıca bu düğüm herhangi bir hataya maruz kalıp aktörler kullanılamaz duruma geldiğinde bunları tekrar çalışır duruma getirmek oldukça zor bir işlem olacaktır.



Şekil 2.2: Bir Bilgisayar Düğümü Üzerinde Yer Alan Aktörler ve Parça Temsili

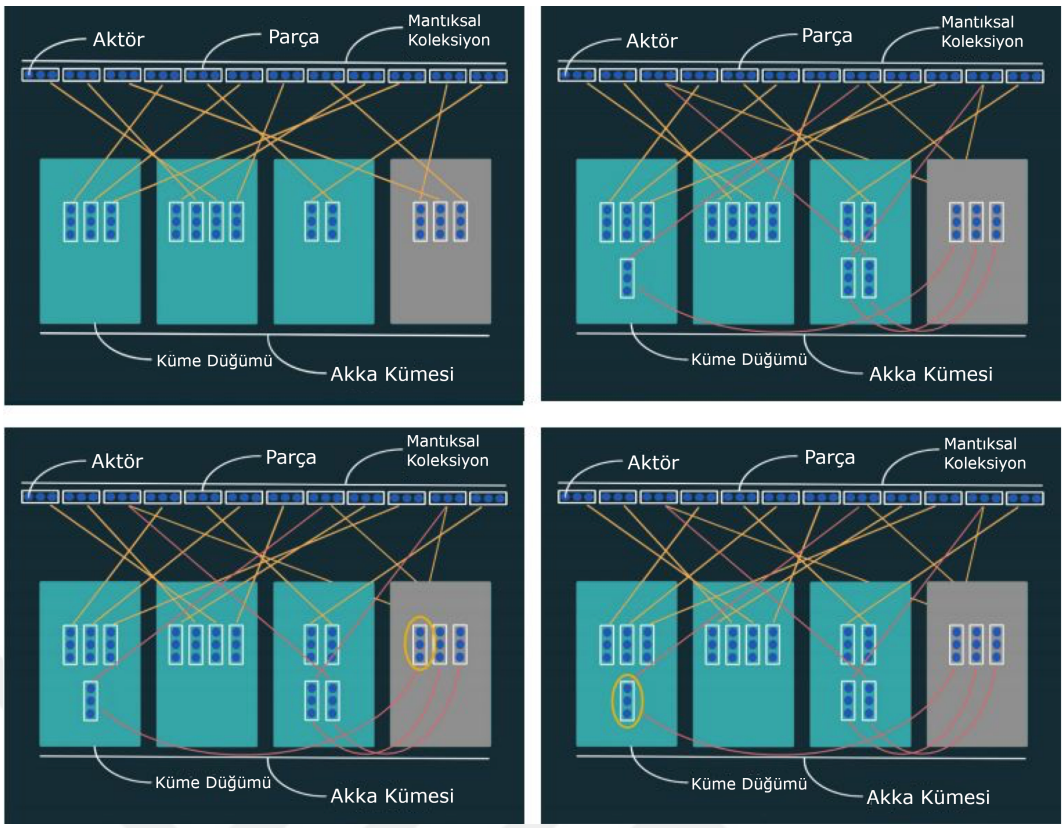
Şekil-2.3'te görüldüğü gibi küme parçalama yöntemi bu çok sayıda aktörün mantıksal parçalara ayrılmasına olanak tanımaktadır. Küme parçalama

yönteminde iyi-tanımlanmış bir strateji ile her parça küme üzerinde düzgünce dağıtılabilmektedir.



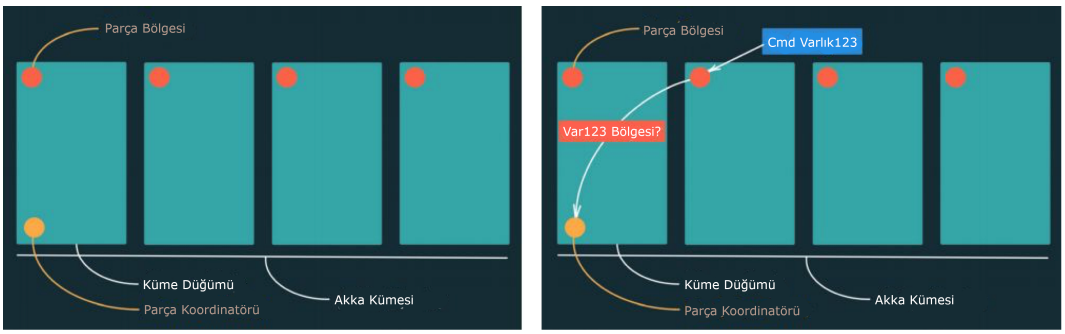
Şekil 2.3: Varlık Aktörü Topluluğunun Küme Parçalama ile Küme Üzerinde Temsili

Şekil-2.4'te her parçanın 4 düğümden oluşan bir küme üzerine dağıtıldığı bir küme parçalama örneği görülmektedir. Burada 4. düğümün çöktüğü bir senaryo oluşturulmuştur. Bu durumda düğümde yer alan parçalar (shards) kümeyi oluşturan diğer düğümler üzerine uygun ve eşit olacak şekilde taşınarak parçalardaki aktörler yeniden canlandırılır. Böylece küme sisteminin çalışması devam eder.



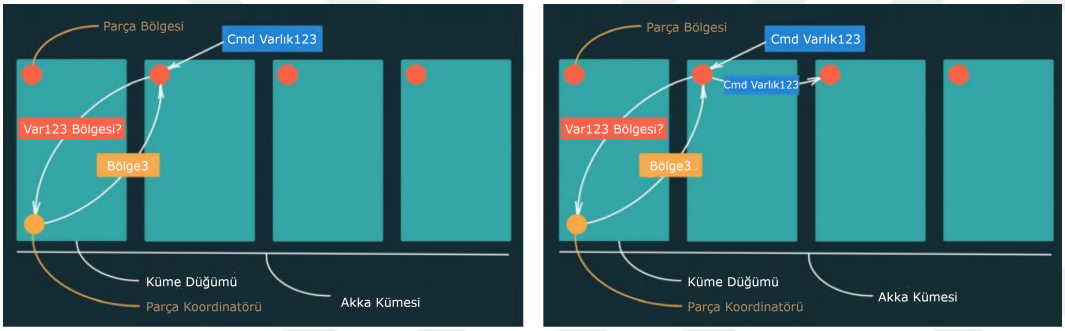
Şekil 2.4: Küme Parçalama Sisteminde Bir Düğümün Çökmesi Durumu

Aşağıdaki Şekil-2.5, Şekil-2.6 ve Şekil-2.7'de küme parçalama örneğinde küme üzerine gelen bir mesajın uygun aktöre nasıl yönlendirildiği anlatılmaktadır. Şekil-2.5'te 4 düğümlü bir küme parçalama senaryosu oluşturulmuştur. Her düğümde birer parça bölgesi aktörü ve tüm kümede yalnızca bir tane bulunabilen parça koordinatörü aktörü görülmektedir. Öncelikle kümeye bir istek geldiğinde bir yük dengeleyici devreye girerek mesajları Round Robin algoritmasına göre kümedeki düğümlere dağıtır. Şekil-2.5'teki örnekte de mesajın ilk olarak 2. düğüme düştüğü görülmektedir. Varlık123 aktörüne gelen mesajı bu aktöre iletmek için öncelikle düğüm-2, düğüm-1'de yer alan parça koordinatörüne Varlık123'ün bölgesini sorar. Hangi parçanın hangi bölgede (düğümde) yer alacağına karar veren aktör koordinatördür.



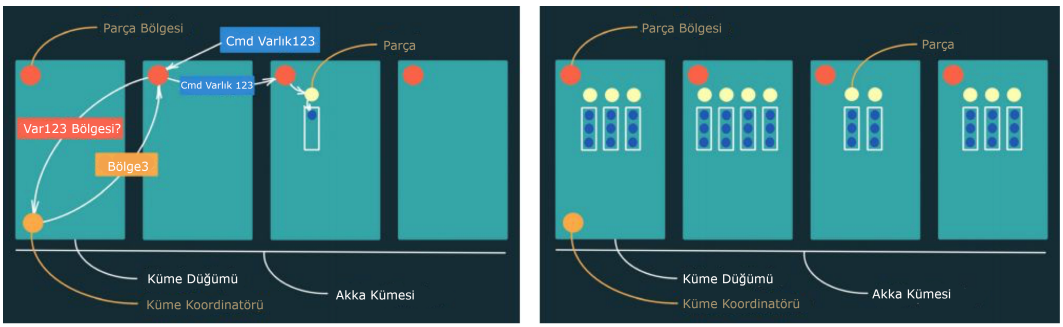
Şekil 2.5: Küme Parçalama Sistemi Üzerinde Mesaj İşletimi-1

Şekil-2.6'da koordinatör Varlık123'ün bölgesinin 3. bölge olduğunu 2. parça bölgesi aktörüne bildirir. Bu durumda da mesaj bölge-2'den bölge-3'e yönlendirilir.



Şekil 2.6: Küme Parçalama Sistemi Üzerinde Mesaj İşletimi-2

Son olarak Şekil-2.7'de parça bölgesi-3 kendisine gelen mesajın bölgesini tekrardan parça koordinatörüne sorarak bölgenin kendi bölgesi olan 3. bölge olduğunu öğrenmiş olur. Daha sonra ilgili varlık aktörü için parçanın ve varlığın bulunmasını sağlayan metotları (extractShardId ve extractEntityId) çalıştırarak mesajı buradaki varlık aktörüne iletir.



Şekil 2.7: Küme Parçalama Sistemi Üzerinde Mesaj İşletimi-3

2.5 Birleşim (Join) Yöntemleri

Birleşim (join) işlemi veritabanı literatüründe önemli bir yere sahiptir ve bir veritabanı sisteminin olmazsa olmaz bir parçasıdır. Literatürde dört temel birleşim algoritması bulunmaktadır. Bunlar iç içe döngülü (nested loop), bağlamalı (bind), sıra-birleştir (sort-merge) ve karma (hash) birleşim (join) yöntemleridir ve yalancı-kod (pseudocode) yapısında sırasıyla Şekil-2.8, 2.9, 2.10 ve 2.11'te yer almaktadır. Her bir algoritmanın bir diğerine üstün geldiği durumlar olabilmektedir.

```
R = execute(q1,ds1)
For each tuple r in R do
     q2' = rewriteTuple(q2,r) // r'deki birleşim anahtarı değerini yeniden yaz
     S = execute(q2',ds2)
     if S is not empty
     then
          result = result + S U r
     end
end
```

Şekil 2.8: İç İçe Döngülü Birleşim (Nested Loop Join) Algoritması

Şekil-2.8'de iç içe döngülü birleşim[7] algoritmasının işleyişi görülmektedir. Agoritmanın ilk adımında q_1 (query 1) sorgusu ds_1 (data source 1) veri kaynağı üzerinde işletilerek R (result) sonucu elde edilir. Daha sonra bu R sonucunun içindeki tüm r (row) satırları sırasıyla işletilerek ikinci q_2 sorgusu her bir r satırında bulunan birleşim anahtarı (join key) değeri kullanılarak q_2' sorgusu olarak yeniden yazılır. Her satır için yeni oluşturulan q_2' sorguları işletilerek S

[7]https://en.wikipedia.org/wiki/Nested_loop_join

sonucu elde edilir. Eğer elde edilen S sonucu boş değilse r satırı ile birleştirilerek ana sonuca eklenir. Bu algoritma en temel ve sıklıkla kullanılan birleşim yöntemidir. Fakat her satır için yeniden sorgu yazılması ve bu sorguların işletilmesi veri kaynağına çok sayıda isteğin atılması demektir. Bu durum ağ üzerinde bir yüklenme (overhead) yaratarak sorgu performansını düşürebilir. Fakat ilk işletilen q_1 sorgusunun seçiciliğinin (selectivity) yüksek, q_2 sorgusunun seçiciliğinin çok düşük olduğu durumlarda yöntemin etkinliği ön plana çıkar. Bu durumda R sonucu az sayıda satırdan (r) oluşur. Dolayısıyla q_2 sorgusu için yeniden yazma sayısı da az olacaktır. Ayrıca q_2 sorgusunun seçiciliği kendi başına düşük olduğundan, birleşim bağlayıcısının (join attribute) r satırındaki değeri kullanılıp q_2' olarak yeniden yazıldığında seçiciliği bir anda yükselir ve sorgu işletim maliyeti q_2'ye oranla oldukça düşer. Bu da diğer birleşim mekanizmalarına göre etkin bir sorgu işletimi sağlar. Genel olarak işletilen sorguların seçiciliğinin yüksek olduğu durumlarda iyi performansa sahiptir. Ancak birleşim yapılacak sorgular seçici olmadığında algoritmanın performansı önemli oranda düşer.

```
R = execute(q1,ds1)
q2' = rewriteTuplesAsFilter(q2,R) //R'deki tüm birleşim anahtarı değerlerini filter
//bloğuna yaz
S = execute(q1',ds2)
if S is not empty then
     result = result + S U R
end
```

Şekil 2.9: Bağlamalı Birleşim (Bind Join) Algoritması

Şekil-2.9'da bağlamalı birleşim algoritmasının işleyişi görülmektedir. Öncelikle q_1 sorgusu ds_1 veri kaynağı üzerinde işletilerek R sonucu elde edilir. Daha sonra R sonucunun içindeki tüm satırlar işlenerek birleşim bağlayıcısı (join attribute) değerleri elde edilir. q_2 sorgusunun birleşim bağlayıcısı elde edilen q_1 sorgu sonucunun birleşim değerleri içerisinde aranacak şekilde bir filter bloğu oluşturularak q_2 sorgusu yeniden yazılır. Yeniden yazılan q_2' sorgusu veri kaynağı ds_2 üzerinde işletilerek S sonucu elde edilir. S sonucu boş değilse R sonucuna eklenerek ana sonuç elde edilir. Bağlamalı birleşim temelinde filter bloğu ile düzenlenmiş nested loop join algoritmasıdır ve ilk olarak

(Haas et al., 1997) tarafından ortaya atılmıştır. Algoritmanın esas etkinliği q_1 sorgusunun seçiciliğinin düşük olduğu sorgularda ortaya çıkar. Bu durumda iç içe döngülü birleşimde ortaya çıkan tüm sonuçlar için q_2' sorgusu yazılması ve tüm q_2' sorgularının ikinci veri kaynağı (ds_2) üzerinde işletilmesi gerekecektir. Bu da hem sorgu işletimi hem de sonuç elde edilmesi adına yüksek bir ağ maliyeti yaratır. Bağlamalı birleşimde ilk sorgudan elde edilen ara sonuçlar ikinci sorgunun filter bloğuna eklenerek q_2' sorgusu elde edildiğinden hem ara sonuçlar tek seferde ağ üzerinden iletilir hem de q_2' sorgusunun seçiciliği artırılmış olur.

```
// Aşama-1 sıralama
sort R on R.a
sort Q on Q.b
// Aşama-2 birleştirme
r = first tuple in R
q = first tuple in Q
while r != EOR and q!= EOR do
     if R.a > Q.b then
          q = next tuple in Q after q
     else if R.a < Q.b then
          r = next tuple in R after r
     else
          put r o q in the output relation
          // r ile eşleşen ileriki satırları sonuca ekle
          q' = next tuple in Q after q
          while q' != EOR and r.a = q'.b do
               put q' o r in the output relation
               q' = next tuple in Q after q'
          // q ile eşleşen ileriki satırları sonuca ekle
          r' = next tuple in R after r
          while r' != EOR and r'.a = q.b do
               put r' o q in the output relation
               r' = next tuple in R after r'
          r = next tuple in R after r
          q = next tuple in Q after q
end
```

Şekil 2.10: Sırala-Birleştir Birleşim (Sort-Merge Join) Algoritması

Şekil-2.10'da Sırala-Birleştir Birleşim[8] algoritmasının işleyişi görülmektedir. Sıralama ve birleştirme olmak üzere iki aşamadan oluşmaktadır. İlk

[8]http://www.dcs.ed.ac.uk/home/tz/phd/thesis/node20.htm

aşamada her iki sonuç da birleşim bağlayıcıları ($R.a$ ve $Q.b$) üzerinden sıralanır. İkinci aşamada her iki sonuç da aynı anda ilerletilerek satır değerleri r ve q elde edilir. Bu iki satırın birleşim bağlayıcılarına karşılık gelen değerleri kıyaslanarak eşit ise r ve q birleştirilerek nihai sonuca dahil edilir. Eşit olma durumunda sonucun değerleri sıralı olduğundan ard arda aynı değerleri içerme olasılığı vardır. Bu durumda daha az kıyaslama yapabilmek adına sırasıyla ikinci sonucun süregelen satırları q' ile r, daha sonrasında da ilk sonucun süregelen satırları r' ile q; birleşim bağlayıcısı değerlerinin eşitliği üzerinden kıyaslanarır ve değerler eşit ise birleştirilerek nihai sonuca eklenir. Birbirinden bağımsız veri kaynaklarından elde edilen ve boyutu büyük sonuçların birleştirilmesinde etkilidir. Algoitmanın en büyük maliyeti ve dezavantajı sıralamadır. Özellikle büyük sonuçlarda bu maliyet daha da artacağından, sonuçların veri kaynaklarından sıralı olarak elde edilmesi durumunda etkinliği oldukça yüksektir.

```
// R ilişkisini karmala
For each tuple r in R do
     put r in bucket no h(r.a)
end
// Q ilişkisini araştır
For each tuple q in Q do
     for each tuple r in bucket no h(q.a) do
          if r.a = q.b then
               put r o q in the output relation
          end
     end
end
```

Şekil 2.11: Karma Birleşim (Hash Join) Algoritması

Şekil-2.11'de basit karma birleşim[9] algoritmasının işleyişi görülmektedir. Öncelikle her iki sonuç arasından daha az kayıt içeren R sonucundaki her bir r satırı birleşim bağlayıcısı üzerinden bir karma fonksiyonuna $h(r.a)$ sokularak üretilen değere göre karma tablosunda eşleşen satırları bulunduran bir kovaya (bucket) koyulur. Daha sonra büyük olan Q sonucu içindeki her bir satır

[9]http://www.dcs.ed.ac.uk/home/tz/phd/thesis/node21.htm

dolaşılarak karma tablosundan $h(q.a)$ değerine denk gelen kovadaki satırlarla birleşim bağlayıcısı değeri üzerinden kıyaslanarak ana sonuç oluşturulur. Basit karma birleşim algoritması R sonucunun belleğe tamamen sığdığı ve birleşim yapılacak sonuçların büyük olduğu durumlarda iyi sonuç verir. Fakat R sonucu belleğe sığmaz ise bu durumda sığmayan bölüm diske yazılarak algoritma parça parça işletilir. Karma tablosunun oluşturulma sayısı, dolayısıyla diske yazma arttıkça algoritmanın performansı düşer. Bunun dışında sorgu sonuçlarına ek karma veri yapıları oluşturulduğundan ve birleşim işlemi veri kaynağı tarafında değil de sorgu işlemcisi tarafında yapıldığından diğer algoritmalara kıyasla daha çok bellek harcanır.

3 İLİŞKİLİ ÇALIŞMALAR

Bu bölümde çoklu depo sistemleri alanında yürütülen güncel litatür çalışmaları ele alınıp incelenmektedir.

3.1 Çoklu Depo Sistemleri (Multistore/Polystore Systems)

Çoklu Depo Sistemleri gevşek-bağımlı, sıkı-bağımlı ve melez olmak üzere üç ana kategori altında toplanmaktadır (Bondiombouy and Valduriez, 2016). Bu ana kategorilerde yer alan sistemler de belli başlı alt birimlerden meydana gelmektedir.

- Gevşek Bağımlı Sistemler: Kendi ortak sorgu dili olan ve çoklu deponun yalnızca bu sorgu dili üzerinden sorgulanabildiği çoklu depo sistemleridir. Ortak sorgu dilinde yazılmış alt sorguların, yere depoların sorgu dillerine dönüşümlerinin yapılmasını gerektirir. Uç kullanıcılar için sorgulama işlemini oldukça kolaylaştırır.
- Sıkı Bağımlı Sistemler: Çoklu depo sisteminin yalnızca yerel depoların sorgu dilleri ile sorgulanabildiği, ortak sorgu dili olmayan sistemlerdir. Yerel depoların sorgu dillerine hakim olmayı gerektirdiğinden, sorgulama nispeten zorlaşır. Bunun aksine sorgu işletimi süreci ise çok daha kolaydır.
- Melez Sistemler: Hem ortak bir sorgu dili ile, hem de depoların yere sorgu dilleri ile sorgulanabilen çoklu depo sistemleridir. Esnek olması dolayısıyla hem gevşek bağımlı hem de sıkı bağımlı sistemlerin olumlu yönlerine sahiptir. Ancak her iki türün de geliştirim özelliklerini barındırdığından mimari aynı ölçüde karmaşıklaşır.

(Tan et al., 2017) çoklu depo sistemlerinde sorgu işleme üzerine bir inceleme (survey) çalışması yapmışlardır. Bilgi bütünleştirme sistemlerinin (information integration systems) araştırmacılar tarafından daha iyi anlaşılabilmesi

için bir çerçeve ve taksonomi sunmuşlardır. Taksonomiye göre bilgi bütünleştirme sistemlerini: federated database sistemleri (tek arayüz - homojen depolar), polyglot sistemler (çok arayüz - homojen depolar), multistore sistemler (tek arayüz - heterojen depolar) ve polystore sistemler (çok arayüz - heterojen depolar) olmak üzere 4 başlık altında sınıflandırmışlardır. Çerçeve; heterojenlik (veri deposu, işleme motoru, sorgu arayüzü), otonomluk (bağlantı, çalıştırma, gelişim), şeffaflık (yer, dönüşüm), esneklik (şema, arayüz, mimari) ve optimallik (birleşim planı, veri yerleşimi) olmak üzere 5 farklı bakış açısından değerlendirme sunmaktadır. Çalışmada BigDAWG (Duggan et al., 2015; Gadepally et al., 2016), CloudMdsQL (Kolev et al., 2016a,b), Myria (Wang et al., 2017a; Halperin et al., 2014) ve Apache Drill (Hausenblas and Nadeau, 2013) araçları tüm bu bakış açılarından değerlendirilmiştir. Sonuca göre sistemlerin birbiri üzerinde bir üstünlüğü gözlemlenmemiş olmakla birlikte, bu sistemlerin özellikleri, bileşenleri ve tasarım ödünleşimleri (trade-off) ortaya konmuştur. Aşağıda sırasıyla bu güncel araçlar ve burada ele alınmayan ESTOCADA ve Forward araçları ile diğer özgün yaklaşımlar incelenmektedir.

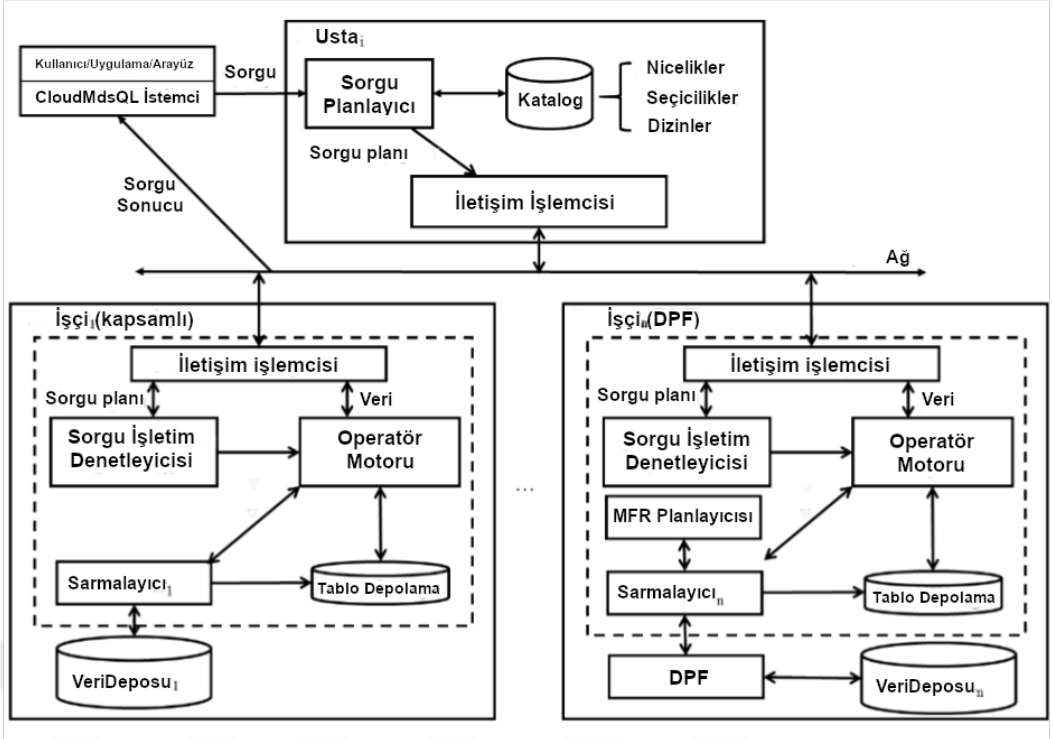
3.1.1 CloudMDSQL

CloudMdsQL; INRIA Enstitüsü, Zenith Bilimsel Veri Yönetimi (INRIA-National Institute for Research in Computer Science and Control, Zenith-Scientific Data Management) araştırma takımı tarafından ve Patrick Valduriez yönetiminde geliştirilmiştir. SQL dili genişletilmiş ve böylece hem SQL hem de veri deposunun özerk sorgu dili ile esnek bir sorgulama yeteneği sunulmuş ve yüksek heterojenlik sağlanmıştır. Ayrıca araç bulut ortamında birden fazla bilgisayar düğümü üzerinde dağıtık olarak çalışabilmekte ve böylece düğümler arasında sorgu planı ve veri değiş tokuşu mümkün olabilmektedir. Mimari olarak modüler bir yapıya sahip olmasına karşın kolayca genişletilebilir değildir.

(Bondiombouy et al., 2015) ilişkisel veri ve HDFS verisinin SQL benzeri bir sorgu dili ile bütünleştirimi üzerine çalışmışlardır. Bunun için ilişkisel veri tabanı Apache Derby'nin sorgu motoru genişletilerek tasarlanan "CloudMdsQL" isimli sorgu dilinin derleyicisi Java dilinde yazılmıştır. Yeni sorgu motoru

SQL sorgularını Derby'nin JDBC sürücüsünü kullanarak işletirken, HDFS üzerine atılan MFR (Map-Filter-Reduce) sorgularını işlemek amacıyla Spark için bir sarmalayıcı (wrapper) geliştirmişlerdir. Yapılan bu sorgu derleyicisi deneysel bir veri kümesi üzerinde geçerlenmiştir. Bu veri kümesinde; yayınlar ve yayınlar ile ilgili anahtar kelimelerin bulunduğu iki tablo verisi ilişkisel veritabanında, bilimsel bir forumdan alınan yapısal olmayan metin verisi ise HDFS üzerinde saklanmıştır. Böylece forumda geçen yapısal olmayan metinler ile ilişkisel yapıda tutulan yayınlar arasında bağlantı kurularak bütünleştirme sağlanmış olacaktır. Belirlenen örnek bir sorgu için aracın sorgu işletim süresi açısından performansı; 3 farklı çalıştırma planı ve farklı HDFS konfigürasyonları üzerinden değerlendirilmiştir.

(Kolev et al., 2016a,b) çoklu depo adı altında farklı veri modellerine sahip depolara (veritabanı) (kullanım durumu için ilişkisel, belge ve çizge) atılacak sorguların yönetimi için bir sistem tasarlamışlardır. Çalışmalarda vurgulanan ana katkılardan birisi çoklu depo sistemini sorgulamak için tasarlanan SQL tabanlı sorgu dilidir. Buna göre çoklu depo sistemini oluşturan her bir veritabanı tasarlanan ortak sorgu diliyle veya depoya özel sorgu diliyle sorgulanabilmektedir. Bir diğer önemli katkı ise çoklu depo sisteme atılan sorguları çalıştırıp sonuçları uygun şekilde bütünleştirip oluşturacak bir sorgu motoru gerçekleştirimidir. Buna göre sorgu motoru tasarlanan "CloudMdsQL" dili söz diziminde sisteme gelen sorguları çözümlemekte, uygun sorgu planını oluşturmakta ve alt sorguları çalıştırılmak üzere ilişkili olduğu veritabanı için yazılmış sarmalayıcı yapılara yönlendirmektedir. Her bir alt sorgu ilişkili olduğu sarmalayıcı aracılığıyla ilgili depo üzerinde çalıştırılmakta ve dönen sonuçlar sorgu çalıştırıcı yapı tarafından birleştirilerek ana sorgu sonucu oluşturulmaktadır. Sorgu motoru bulut ortamında çalışmaktadır ve Şekil-3.1'te görüldüğü üzere tam dağıtık mimari yapıdadır. Böylece düğümler birbirleriyle kod (sorgu planı) ve veri alışverişi yapabilmektedirler. Her düğüm (node) bir usta (master) ve bir işçiden (worker) oluşmakta ve birbirleriyle bir iletişim işlemcisi aracılığıyla haberleşmektedirler. Düğümler sorgu planlama için bir usta, sorgu işletimi için ise bir ya da daha fazla işçi içermektedirler.



Şekil 3.1: CloudMDSQL Sorgu Motoru Mimarisi

Çoklu depo bütünleştirme için ön çalışmaları ve ilk gerçekleştirimi yapılan "CloudMdsQL" sorgu dili ve çoklu depo sistemi fikrini (Kolev et al., 2016d) hayata geçirmiştir. Bu çalışmada tasarlanan sorgu dili ve bu dile özel geliştirilen sorgu motoru tüm detayları ve işleyişiyle ele alınmıştır. Öncelikle veri modellerine ve sorgu dilindeki temel yapılara değinilmiştir. Daha sonra sorgu motorunun yapısı ve temel bileşenleri anlatılmıştır. Dağıtık halde master-slave yapıda çalışan sorgu motorundaki düğümler veri ve sorgu planı değiş tokuşu yapabilmektedir ve sorgu işletimi yöntemlerinden bağlı birleşimi (bind join) esas almaktadır. Sorgu dilinin özgünlüğü ise hem SQL sorgularını hem de veri depolarına (data stores) özgü sorguları bir arada çalıştırabilmesidir. Python dilinde yazılan sorgu derleyicisi depolar arası bütünleştirimi Python'da yer alan "geçici soyut tablo ifadeleri" (named-table-expression) yardımı ile sağlamaktadır. Sorgu işletiminde öncelikle sorgu yorumlanarak ağaç yapısında öncül sorgu işletim planı oluşturulmaktadır. Maliyet modeline (cost-model) veya üst veriye dayanarak alternatif sorgu planları oluşturulup en iyisi

seçilmekte ve ilk işçi (worker) düğüme iletilerek bağlı birleşim yöntemiyle ve sırasıyla ilgili depoların sarmalayıcı fonksiyonları kullanılarak işletilmektedir. Tasarlanan sorgu dilinin ve sorgu motorunun çalıştığını göstermek için 3 adet deponun bulunduğu bir kullanım durumu yaratılmıştır. Buna göre ilişkisel veritabanında bilim insanlarının bilgileri, belge veritabanında yayın ve yayınlar ile ilgili yorum bilgileri, çizge veritabanında ise bilim insanlarının birbirleriyle olan arkadaşlık bilgileri tutulmaktadır. Değerlendirme (evaluation) kısmında ayrıca sorgu motorunun gerçekleştirim yapısı, veri kümelerinin içerdiği veri miktarı ve seçicilik değerleri, değerlendirme ortamının kurulum ayarları da dahil olmak üzere tüm ayrıntılar açıkça anlatılmıştır. Eniyileştirmelerin etkisini göstermek için 5 adet sorgu ve her sorgu için de 3 farklı sorgu işletim planı oluşturulmuş ve sonuçlar sorgu işletim süreleri cinsinden grafiksel olarak gösterilip yorumlanmıştır.

(Bondiombouy et al., 2016; Kolev et al., 2016d) genel sorgu dili çalışmasını genişletilerek sistemi MFR (Map-Filter-Reduce) operasyonlarını da destekler hale getirmişlerdir. CloudMdsQL sorgu dili ve bu sorgu dilini çalıştıran sorgu motorundaki yapılar aracı-sarmalayıcı (mediator-wrapper) prensibine göre çalışmaktadır. Buna göre sorgu işlemcisi; aracı, sorgu motoru ve depolar arasında iletişimi kuran ve sarmalayıcı ise; sorgu dönüşümlerini gerçekleştiren yapılardır. Genel sorgu motoru yapısı tanıtıldıktan sonra SQL-MFR dönüşümlerinden bahsedilmiştir. Ayrıca etkin bir sorgu işletim yöntemi olan bağlı birleştirim yönteminin tasarlanan genişletilmiş sorgu dilinde ve aracında ne şekilde kullanıldığı anlatılmıştır. Daha sonra ilişkisel, belge ve HDFS depolarını kapsayan bilimsel yayınlar ile ilgili bir kullanım durumu ve bununla ilgili bir sorgu tasarlanıp bu sorgunun örnek işletimi anlatılmıştır. Son olarak genişletilen sorgu dilinin ve tasarlanan sistemin etkisini göstermek için bu 3 veri kümesini içeren bir değerlendirme ortamı kurulmuştur. Ortamın tüm özellikleri ve veri kümelerindeki seçicilik (selectivity) değerleri ve veri miktarları açıklanmıştır. Tasarlanan sorgular; veri kümelerinde farklı seçicilik değerleri kullanılarak, farklı makine küme (cluster) konfigürasyonları üzerinde, ayrıca bağlı birleşim kullanılarak ve kullanılmayarak çalıştırıldığında elde edilen

sonuçlar ortaya konmuş ve grafiksel olarak yorumlanmıştır.

(Kolev et al., 2016c) CloudMdsQL aracı ile çoklu depo kullanım durumu üzerinden bir kıyaslamalı değerlendirme yapmışlardır. Öncelikle CloudMdsQL mimarisine ve işleyişine değinilmiş, buradaki sorgu dili, bind join yöntemi ve sorgu motoru anlatılmıştır. Daha sonra kıyaslama ve performans değerlendirme bölümünde bind joinler ve yerel sorgu desteği sınanmıştır. Deneysel değerlendirme 5 farklı veri deposu ile TPC-H benchmarkı yüklenerek gerçekleştirilmiştir. Sonuç olarak da özellikle bind join kullanılan çapraz sorgu durumlarında sistemin başarısı ortaya konmuş ve yerel sorguların değerlendirilmesinde de iyi sonuçlar alınmıştır.

(Kolev et al., 2018) çalışmalarında heterojen veri depoları üzerinde paralel çoklu depo sorguları çalıştırmışlardır. Önceki çalışmalarında tasarladıkları CloudMdsQL sorgu dilini kullanıp, sorgu paralelliğini ve dağıtıklığını LeanXcale veritabanının sorgu motoru ile sağlamışlardır. Burada sorgu motorunun işçi thread bazlı ölçeklenebilirliği, kullanılan veri deposunun (çalışmada MongoDB için gerçekleştirimi yapılmış) parçalamaya (sharding) dayanan saklama ölçeklenebilirliği ile desteklenerek bağımsız bir yatay ölçeklenebilirlik elde edilmiştir. Son olarak bağlı birleşimin (bind join) paralel sorgu işletimine uygun şekilde çalışabildiği ve performansı artırdığı ortaya konmuştur. (Kranas et al., 2021) bu çalışmayı genişleterek Spark aracı ile çoklu depoyu genişleterek alt sorgu olarak MFR (map/filter/reduce) işlemcilerini de kabul etmektedir. Spark kullanılarak sorgu işleme ve ara sonuçların elde edimi (bağlı birleşim ile) işlemlerini paralel olarak gerçekleştirebilmektedir. Bu kapsamda sorgu işlemcisi LeanXcale ile Spark aracı bağlı birleşim kullanılarak ve kullanılmadan kıyaslanmıştır. Buna göre birleşim işlemlerini paralel olarak yapmanın performansı artırdığı ortaya konmuştur. Değerlendirmeye ek olarak paralel çoklu depo mimarileri farklı bakış açılarından bir özellik tablosu üzerinden kıyaslanmıştır.

3.1.2 BigDAWG

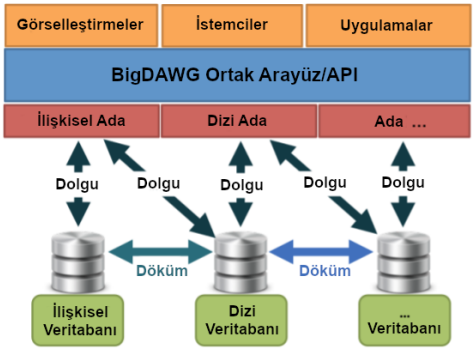
BigDAWG; MIT üniversitesi ve ISTC (Intel Science and Technology Center for Big Data) çatısı altında, Michael Stonebraker ve Sam Madden'ın ekibi tarafından geliştirilmiştir. Bilgi adaları soyutlaması ile heterojenliği, optimizer ve migrator modülleri ile sorgu planı ve veri yerleşim optimizasyonu dolayısıyla da optimalliği yüksek seviyede sağlamaktadır. SCOPE ve CAST anahtar kelimeleriyle adalar arası sorgu işletimi mümkün olmaktadır. Sistemde dağıtık ve paralel sorgu işleme yeteneğinden bahsedilmemektedir, ayrıca sistem modülleri birbirine bağımlı olduğundan kolayca genişletilebilir durumda değildir.

(Duggan et al., 2015) birden fazla veri modelini kapsayan bir bilgi yönetimi ihtiyacına vurgu yaparak yeni bir birleştirilmiş veritabanı (federated database) bakış açısı olan çoklu depo (polystore) yaklaşımını ortaya atmışlardır. Intel Science ve Technology Center for Big Data gruplarıyla ortak çalışılarak, MIMIC-II (Saeed et al., 2011) hastane veri kümesi üzerinde farklı veri modellerini kapsayan bir kullanım durumu oluşturulmuş ve tasarlanan BigDAWG isimli çoklu depo sistemi prototipi bu kullanım durumu üzerinde denenmiştir. Bu yeni yaklaşımda üç hedef üzerine odaklanılmıştır. Bunlardan ilki saklanacak nesneler için "yer şeffaflığı"dır (location transparency) ki böylece kullanıcılar birden çok veri yönetim sistemini kapsayan bildirimsel (declerative) sorgular atabilmektedir. Çalışma kapsamında ortaya atılan "bilginin adası" (island of information) soyutlaması tek bir sorgu dili ile erişilen saklama motorlarını (storage engine) ifade etmektedir. İkinci hedef "anlamsal bütünlük"tür (semantic completeness) ve öyle ki yeni bir saklama motoru çoklu depoya eklendiğinde, bu saklama motorlarıyla ilgili herhangi bir işlevsellik kaybedilmeyecektir. Üçüncü hedefte ise kullanıcılar belirli bir arka-uçta (back-end) saklanan nesnelere farklı adalar üzerinden erişebileceklerdir. Örneğin; fiziksel olarak belirli bir arka uçta tutulan bir veriye hem dizi (array) hem de ilişkisel (relational) adalar üzerinden erişilebilecektir. Çoklu depo içerisindeki bir ada yer şeffaflığını, her saklama motoru için yazılan bir dolgu (shim) ile

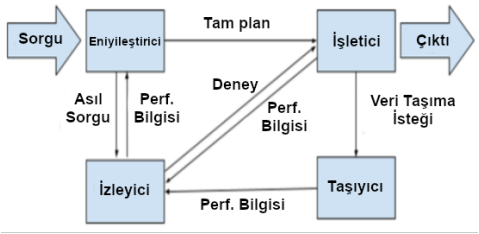
desteklemektedir. Çalışma kapsamında tasarlanan sorgu mekanizmasında ilgili alt sorgunun çalıştırılacağı ada SCOPE anahtar kelimesi ile belirlenmekte ve bir başka alt sorguyla ilişkili birleşim (join) operasyonlarının gerçekleştirilmesi için ise CAST anahtar kelimesi kullanılarak arka-uçlar arasında nesnelerin kopyalanması sağlanmaktadır. Tasarlanan sistemdeki özgün yanlardan biri de şemalar hakkında bir üst-veriye sahip olmayı gerektirmeden kara kutu (black box) yaklaşımıyla sorguların işletileceği arka-uçların belirlenmesi ve sorgu eniyileştirmesinin sağlanmasıdır. Bunun için kara-kutu yaklaşımında üç farklı çalışma modu ile profilleme yapılmaktadır. Eğitim (Training) modunda sorgu tüm saklama motorları üzerinde çalıştırılıp süreleri kaydedilmektedir. Eniyileşmiş (Optimized) modda sorgu en kısa süreyi veren saklama motoru üzerinde işletilmektedir. İyimser (Opportunistic) modda ise yeni saklama motorlarının kaynakları uygun duruma geldiğinde saklanan alt-sorgular bunlar üzerinde işletilmektedir. Böylece kara-kutu yaklaşımıyla sistem üzerinde alt-sorgu bazında dinamik profilleme yapılarak, her alt sorgunun işletileceği en uygun saklama motorunun belirlenmesi sağlanmış olmaktadır.

(Gadepally et al., 2016) bu çalışmayı geliştirip ilerleterek BigDAWG (Duggan et al., 2015; Dziedzic et al., 2015; Gadepally et al., 2016; She et al., 2016) çoklu depo sisteminin mimari yapısını açıklamışlardır. Ayrıca sistemin başlangıç aşamasındaki performans değerlendirmesi de sunulmuştur. Çoklu depo sisteminin genel mimarisi Şekil-3.2'da görülmektedir. Buna göre sistem veritabanı ve saklama motorları (database and storage engines), adalar (islands), ortak arayüz (middleware) ve API, uygulamalar (applications) olmak üzere 4 katmandan oluşmaktadır. Ana katman BigDAWG katmanı olmakla birlikte yazarlar çalışmada bu 4 katmanı sırasıyla açıklamışlardır.

İlk olarak veritabanı ve depolama motorları katmanında BigDAWG'ın altyapısındaki çoklu depo ortamından bahsedilmiştir. Çoklu depo ortamının gerekliliğine vurgu yapılmış ve bir kullanım durumu örneği verilmiştir. İkinci sıradaki katman adalardır (islands). Her adanın bir veri modeli, sorgu dili veya operatör kümesi ve bir ya da daha fazla veritabanı motoru bulunmaktadır. Bu sayede adalar kullanıcı için bir soyutluk sunarak, anlamsal bütünlük



Şekil 3.2: BigDAWG Çoklu Depo Sistemi Mimarisi (Gadepally et al., 2016)



Şekil 3.3: BigDAWG Ara Katmanı (Gadepally et al., 2016)

(semantic completeness) ve yer şeffaflığı (location transparency) arasında seçim yapabilmesini sağlar. Kullanıcı scope operatörünü kullanarak sorgunun çalışacağı adayı belirtir. Üçüncü katman BigDAWG katmanıdır ve bu katman da kendi içerisinde birkaç bileşenden oluşmaktadır. Bu katmanın ana bileşeni BigDAWG ortak arayüz katmanıdır (middleware). Ortak arayüz katmanı; sorguların karşılanması, sorgu planlama, etkin işletim stratejilerini belirleme, önceki sorguların tarihçesini ve performansını tutma gibi işlerden sorumludur. Şekil-3.3'de yapısı görülen ara katmanın; sorgu eniyileştirme (Eniyileştirici) (Gupta et al., 2016), performans izleme (İzleyici) (Chen et al., 2016), veri taşıyıcı (Taşıyıcı) (Dziedzic et al., 2016) ve sorgu işletimi (İşletici) (She et al., 2016) olmak üzere 4 modülü bulunmaktadır. Sisteme gelen sorgu, Eniyileştirici tarafından ayrıştırılarak bir sorgu planı ağacı kümesi yaratılır. İzleyici bu sorgu planı ağacı kümesini alarak varolan performans verisine göre en uygun ağacı belirler. Çalıştırıcı belirlenen sorgu planı ağacını alarak elindeki verilere göre en uygun yöntemi belirleyerek sorguyu çalıştırır. Ayrıca Çalıştırıcı, Taşıyıcıyı kullanarak sorgu motorları ve adalar arasında nesnelerin taşınmasını sağlayabilir. BigDAWG katmanının diğer bileşeni BigDAWG API'dir ve sorguların işletimi için bir arayüz sunar. Sunucu ve istemci bileşenlerinden oluşan API'nin sunucu bileşenleri veritabanlarını birbirine bağlamak için dolgu (shim) denen yapıları kullanır. Dolgular bir adanın yerel sorgu dilinden veritabanının

doğal diline dönüşümü sağlamaktadır. Scope anahtar kelimesiyle sorgunun çalıştırılacağı ada belirlenirken Cast anahtar kelimesiyle de adalar arasında veri dönüşümü gerçekleştirilir. BigDAWG katmanının son bileşeni Çoklu Depo Sorgularıdır. Sorgular genellikle birden fazla adayı kapsadığından etkin bir sorgu işletimi kritik olmaktadır. Sorgular öncelikle eğitim (training) modunda çalıştırılarak mümkün en iyi sorgu planları oluşturulur. Daha sonra üretim (production) aşamasında ise en iyiye yakın olan sorgu planı seçilmeye çalışarak hızlı çalıştırma amaçlanır. Son katman Uygulama ve Görselleştirme katmanıdır. Uygulamalar, istemciler ve görselleştirme araçları, API ve ara katman üzerinden çoklu depo sistemini kullanırlar. Ada soyutlaması kullanıcılara kendi uygulamalarını en çok ilgilendiren veri deposunu seçip faydalanmasına olanak tanımaktadır.

(Gadepally et al., 2017) "BigDAWG Version 0.1" isimli çalışmalarında BigDAWG mimarisini ve çoklu depo (polystore) tanımını netleştirerek sistemin ilk versiyonunu tanıtmışlardır. "Herkese uyan tek bir beden yok" (no one size does not fit all) sloganıyla beraber çoklu depo sistemlerin özelliklerinden bahsetmişlerdir. Buna göre bir çoklu depo sisteminin heterojen veritabanı motorlarından ve çok sayıda programlama dillerinden oluşan ve bunların bütünleşik olarak yönetildiği bir veritabanı yönetim sistemi olduğu anlatılmıştır. Daha sonra sırasıyla BigDAWG sisteminin mimarisi tanımlanmış, içindeki bileşenler ve işlevleri tek tek açıklanmış ve son olarak da BigDAWG API ve BQL (BigDAWG Query Language) dilinin yapıları tanıtılmıştır.

(Yu et al., 2017) yapmış oldukları çalışmada BigDAWG polystore sistemine Vertica ve MYSQL veritabanı motorlarının bütünleştirilmesini ve sistemin TPC-H benchmarkı ile değerlendirilmesini ortaya koymuşlardır. Değerlendirmede PostgreSQL ile Vertica veritabanları kullanılmış, hem tek başlarına hem de BigDAWG üzerinden çalıştırılmışlardır. Öncelikle tekil sorgu performansı çözümlemeleri, daha sonra da çapraz bütünleşik sorgu çözümlemeleri yapılarak sonuçlar ortaya konmuştur. Sonuç olarak BigDAWG'ın tekil sorgularda iyi çalıştığı ve ek bir yük getirmediği, çapraz bütünleşik bazı sorgularda da PostgreSQL ve Vertica'dan daha iyi sonuçlar ortaya koyduğu gözlenmiştir.

3.1.3 Myria

Myria (Wang et al., 2017a; Halperin et al., 2014); Washington Database Group üyeleri tarafından Magdalena Balazinska ve Dan Suciu yönetimi ile danışmanlığında geliştirilmiştir. Bileşik veri analitiği sistemi (federated data analytics system) olarak tasarlanmış ve soyutlamalarından biri olan MyriaL dili ile karmaşık veri analitiğini ifade etmeyi kolaylaştırmaktadır. En önemli soyutlaması ise RACO (Relational Algebra Compiler) adını verdikleri ara katmandır. Aracın heterojenlik özelliği yüksek olmakla birlikte; optimallik açısından da depolar arası veri taşıma, dönüşüm ve alışveriş oldukça kolay ve rahattır. Araç dağıtık çalışmayı ve ölçeklenebilirliği desteklemektedir. En önemli katkısı klasik hash join yönteminden farklı olarak büyük veri analitiğine uygun, veri çarpıklığını giderip veri karıştırma (data shuffling) maliyetini düşürecek yeni bir paralel birleşim mekanizmasıdır. Birleşim algoritması ile ilgili çalışmalarında (Chu et al., 2015) Hypercube shuffle algoritması ile Tributary join olarak adlandırdıkları birleşim mekanizmasını birlikte kullanarak oldukça etkin sonuçlar elde etmektedirler. Aracın genişletilebilmesi için desteklenmeyen veri depoları için yeni dönüşüm kuralları yazılması gerekmektedir, bunun dışında Python tabanlı olması sebebiyle thread yönetimi ve paralellik kısıtlı olmaktadır.

3.1.4 Apache Drill

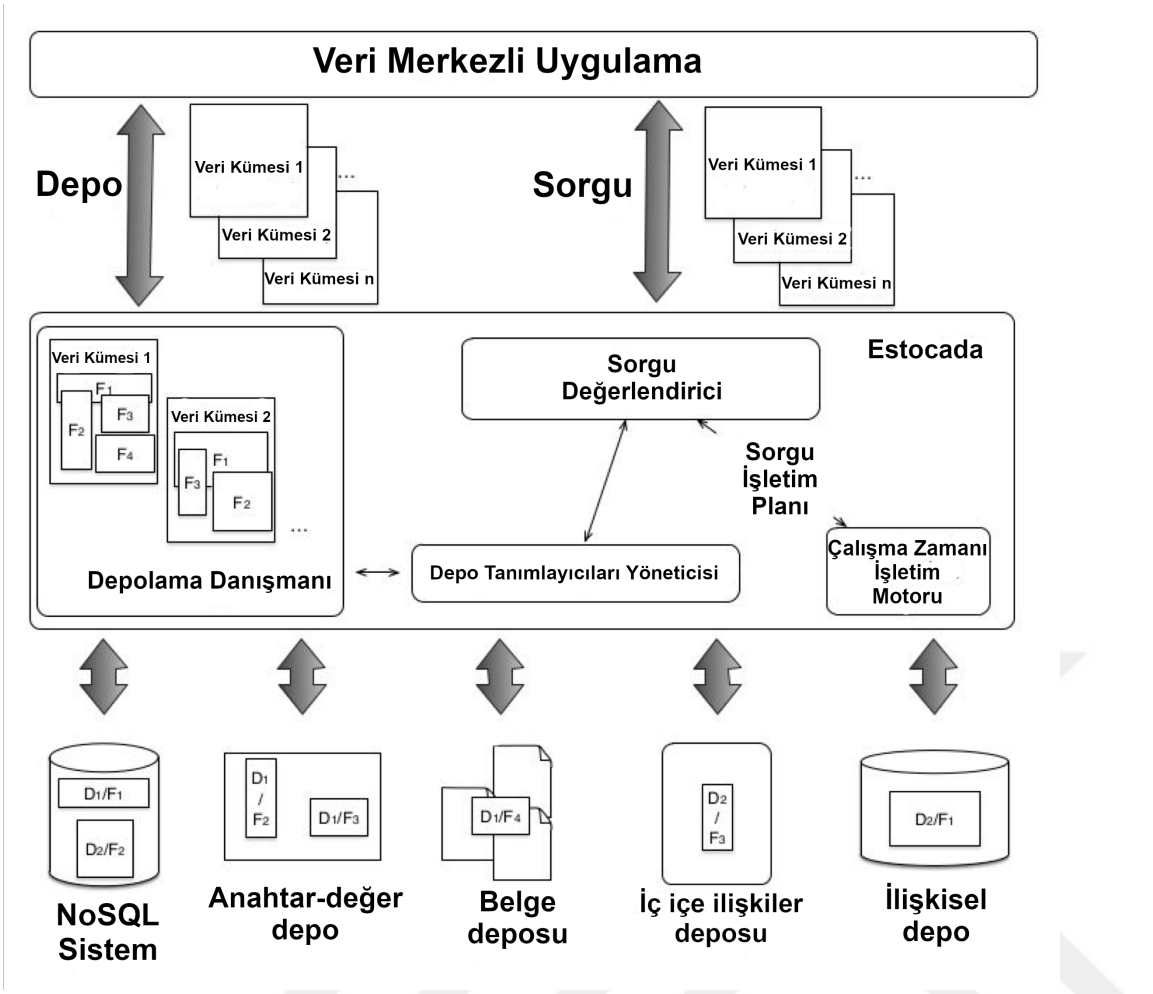
Apache Drill (Hausenblas and Nadeau, 2013) aracı MapR endüstriyel veri analitik platformu desteğinde ve Michael Hausenblas önderliğinde Apache Software Foundation tarafından geliştirilmiştir. Veri analitiğine odaklı; tam anlamıyla dağıtık ve büyük ölçüde paralel bir sorgu motorudur ve bileşenlerinin birçoğu genişletilebilir (pluggable) özelliktedir. Aracın güçlü olduğu özellikleri; sütun tabanlı bellek içi veri modeli ve dinamik şema keşfi yapabilen Apache Calcite tabanlı tam dağıtık ve paralel sorgu motorudur. En önemli tasarımsal yanı ise Drillbit adı verilen ve Hadoop kümesi (cluster) üzerinde çalışabilen daemon servis yapısıdır ve sorgu motorunun tüm özelliklerine sahiptir. Sorgu

işleme sürecinde sorgular parçalı olarak işletilerek sonuçlar yaprak çalıştırma motoru (leaf execution engine) tarafından elde edilir. Elde edilen sonuçlar ebeveyn (parent) çalıştırma motoru (parent execution engine) tarafından birleştirilir. Sistem SQL tabanlı olup belli başlı depoları desteklemektedir. Dolayısıyla bu depolarda sorguların çalışabilmesi için "SQL - yerel sorgu" dönüşümlerinin yazılması gerekmektedir. Drill'in desteklemediği bir depo sisteme ekleneceği zaman yeni bir SQL - yerel sorgu dönüşümü yazılması gerekir ve bu da sistemi heterojenlik bakımından oldukça olumsuz etkilemektedir.

3.1.5 ESTOCADA

Bir diğer çalışmada (Bugiotti et al., 2015a) ölçeklenebilir ve kendi kendini düzenleyebilen bir çoklu depo sistemi tasarlamışlardır. ESTOCADA (Bugiotti et al., 2015a,b, 2016) ismini verdikleri ve altyapısında birçok veri modelini destekleyebilen bu çoklu depo sisteminin temelinde veri bütünleştirme yöntemlerinden LAV (local as view) ve view tabanlı yeniden yazma (view-based rewriting) yatmaktadır. Sistemin mimarisi Şekil-3.4'te görülmektedir.

Yerel sorgu işletimi, alt sorgu sonuçları toplanarak oluşturulan bir alt veri kümesi (view) üzerinde yapılmaktadır. Ayrıca birleşik (conjunctive) sorgularda ve/veya sorgu planı oluşturulurken çalıştırılacak olan sorgu, çalıştırılacağı deponun diline ve kısıtlarına uydurularak yeniden düzenlenmektedir (reformulation under constraints). Sistemde veri kümesi parçalama (dataset fragmentation) yöntemi uygulanarak ölçeklenebilirlik sağlanmaktadır. Yerel depoların performansı sorgulama ve saklama olmak üzere iki kısıta göre değerlendirilip bir sorgu için en iyi çalıştırma planı oluşturulmaktadır. Zaman içerisinde bu planlar ve çalışma süreleri kaydedilip, gerektiğinde sistem tarafından ilgili sorgu parçasının verisinin bir başka depoya yüklenmesi (örneğin verinin ilişkisel veritabanından alınıp sütun veritabanına taşınması) işlemi gerçekleştirilmektedir. Depo düzenlemenin (storage tuning) temelinde ise görüntü seçme (view selection) yatmaktadır. Böylece sistemin kendi kendini düzenleme yeteneğini kazanmasının yanında ölçeklenebilirliği de artmış olmaktadır. (Alotaibi et al., 2019) ESTOCADA aracını MIMIC-III gerçek zamanlı



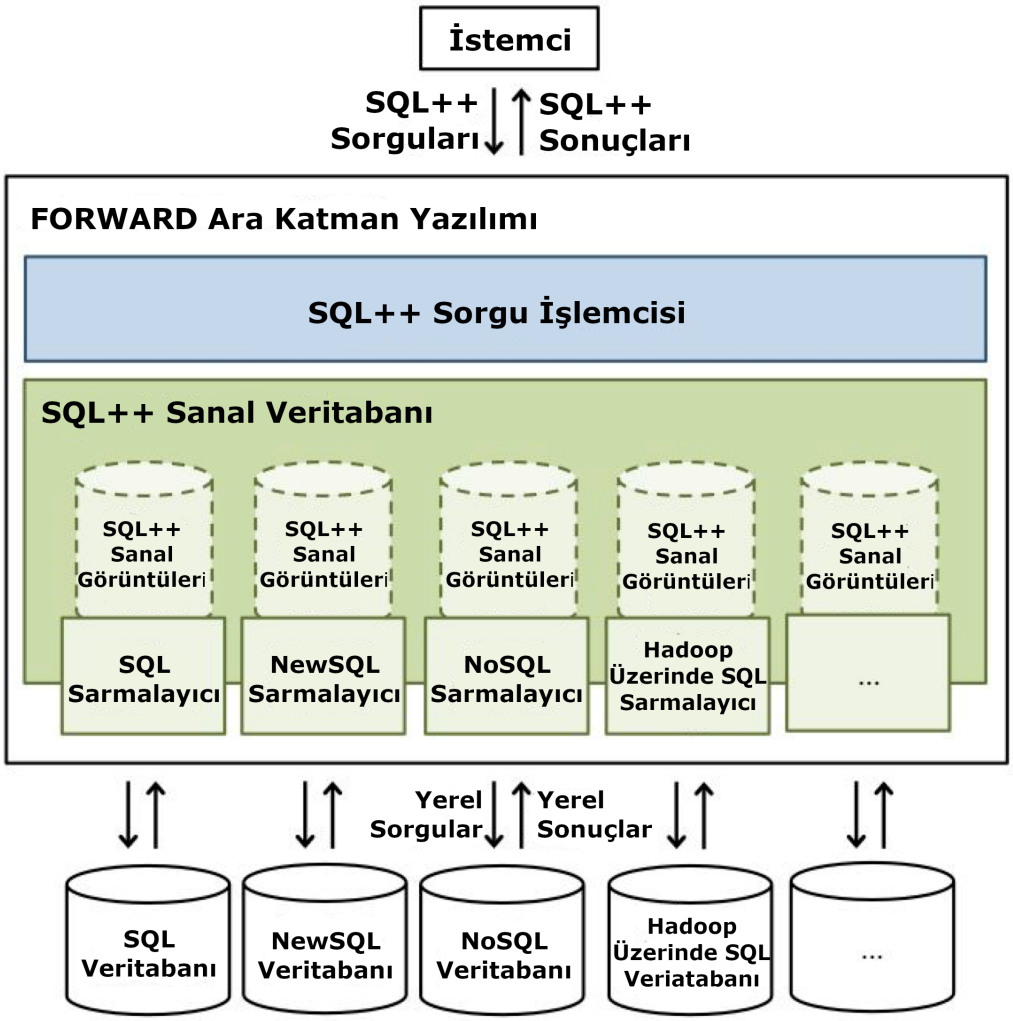
Şekil 3.4: ESTOCADA Mimarisi (Bugiotti et al., 2015a)

veri kümesini kullanarak oluşturdukları test ortamında ve test sorgularıyla tekil ve çoklu depo performansları üzerinden değerlendirmişlerdir. (Alotaibi et al., 2020) ise yapmış oldukları demo yayını çalışması ile aracın arayüzünü ve kullanımını tanıtmışlardır.

3.1.6 FORWARD

(Ong et al., 2014b) çalışmalarında birbirinden farklı modellere ve dil yapılarına sahip NoSQL, NewSQL ve SQL-on-Hadoop veritabanlarını SQL++ (Ong et al., 2014b,a,c; Papakonstantinou, 2016) dilinin birleştirici yönlerini kullanarak birbiriyle kıyaslamışlardır. Çalışmalar farklı yapıdaki veritabanlarının yetenek bakımından birbiriyle kıyaslanması için bir inceleme

(survey) niteliğinde olup bu kıyaslamanın yapılabilmesi için veritabanlarını bütünleştiren bir yapı gereklidir. Yazarlar bu bütünleştirme için gerçekleştirmiş oldukları FORWARD (Ong et al., 2014b,a,c; Papakonstantinou, 2016) isimli ara-katman sorgu işlemcisini kullanmışlardır. SQL++ da FORWARD tarafından kullanılmakta ve birleştirici bir veri modeli ve yarı-yapısal sorgu dili olarak işlev görmektedir. Şekil-3.5'da FORWARD sorgu işlemcisinin yapısı görülmektedir.



Şekil 3.5: FORWARD Mimarisi

Kavramsal olarak her veritabanı istemciye SQL++ kaynağı olarak görülmektedir. İstemci bir veya birden fazla veri kaynağı üzerine SQL++ sorgusu yollamakta ve FORWARD sorgu işlemcisi de bu sorgunun çalışması için

bir ya da birden fazla veritabanının yerel sorgu diline dönüşümü gerçekleştirmektedir. FORWARD sorgu sonuçlarını da veritabanları ile SQL++ arasındaki uyumsuzlukları gidererek gerektiğinde bir ya da daha fazla veri modelini kapsayacak şekilde bütünleşik olarak birleştirmektedir. Yazarlar FORWARD arakatman sorgu işlemcisinin detaylarını yayınlayan bir çalışma yapmamışlardır. Bunun yerine bir kullanım durumu üzerinden işlemcinin sorguyu işleyiş biçimi açıklanmıştır (Ong et al., 2014c).

3.1.7 Diğer Özgün Yaklaşımlar

(Idreos et al., 2017) gelişimsel veri sistemleri isimli yeni bir kavram ortaya atmışlardır. Bu kavrama göre sisteme yeni veriler ve sorgular geldikçe sistem gelişerek o anki iş yükü karakteristiğine en uygun hale gelmektedir. Genetik algoritma kullanılarak veri ve sorgular evrimleştirilip dönüştürülerek yeni kullanılacak yapıya uygun hâle getirilmektedir. Sistem temelde saklama motoru, işletim motoru ve eniyileştirici olmak üzere 3 katmandan oluşmaktadır. Saklama motorunda sorguların atılmadığı soğuk veriler, ve sorgulanan sıcak veriler yer almaktadır. Bu sıcak veriler de saklama modüllerinde aktif ve olası mutasyonlar yapılacak şekilde tutulurlar. Çalıştırma motorunda da mutasyon motoru ve erişim yöntemleri yer alır. Erişim yöntemleri de saklama modüllerine benzer mutasyonları içerir. Eniyileştirici modülde ise sorgu planları, gelişim takvimi ve veri istatistiki bilgileri yer alır. Sistemin ve ortaya atılan kavramların doğruluğu sistemin sınandığı anahtar-değer ve sütun veritabanları arasında uygun dönüşümler yapılarak elde edilen sonuçlar ile ortaya konmuştur.

(Gadepally et al., 2015) birleşmeli diziler (associative arrays) yaklaşımını farklı tipteki veritabanlarını ve saklama motorlarını entegre etmek için kullanmışlardır. Bunun için D4M isimli Dynamic Distributed Dimensional Data Model aracını ortaya geliştirip ilk veritabanı motoru desteği olarak da SciDB bağlantısını sağlamışlardır. (Kepner et al., 2016) yapmış oldukları çalışmada BigDAWG çoklu depo sisteminin farklı tipteki veri kaynaklarını birleştirme yaklaşımından yola çıkarak, bu kaynaklar arasında veri taşımayı ve sorgu işletimini kolaylaştırmanın bir yolunu düşünmüşlerdir. Buna göre birleşmeli di-

ziler (associative arrays) yaklaşımının farklı tipteki veritabanlarının içerisinde yatan matematiğe ortak bir yaklaşım sağladığını ortaya atmışlardır. SQL veri modelinin içindeki yapıların birleşmeli diziler yaklaşımı açısından nasıl tanımlanabileceği ortaya konmuş ve birleşmeli dizilerin çoklu veri depolarındaki operasyonların optimizasyonu için matematiksel bir model sağlayabileceğini göstermişlerdir.

(Maccioni et al., 2016; Maccioni and Torlone, 2018) ortaya koydukları QUEPA isimli araç ile bir polystore ortamında artırma (augmentation) yöntemiyle sorgulama kolaylığı sağlamaktadırlar. Bu araç herhangi bir ortak veri modeli ve konfigürasyon ihtiyacı doğurmadan, doğrudan tak-çalıştır modda kolaylıkla bütünleşip çalışabilmektedir. İlk sorgulamadan sonra kullanıcı olası sorgu kombinasyonlarına (guided-search) yönlendirilmektedir.

(Vajantri et al., 2017) çalışmalarında Apache Calcite[10] (Begoli et al., 2018) aracının çoklu depo ortamı için uygulanabilirlik ve performansını ortaya koymuşlardır. Calcite, SQL sorgularını çalıştırabilen ve ilişkisel cebir üzerinde dönüşümler yapabilen bir sorgu işlemcisidir. Calcite, ilişkisel - SQL Server, dizi - TileDB ve metin - Elasticsearch veritabanları üzerinde, veritabanlarının kendi yerel işlemcileri ile sınanmıştır. Ayrıca Apache Drill aracıyla performans kıyaslaması yapılmıştır. Apache Drill[11] (Hausenblas and Nadeau, 2013) sorguları desteklediği birçok veritabanı üzerinde işleyebilen, birleşim işlemlerini gerçekleştirebilen SQL tabanlı bir birleştirilmiş (federated) sorgu sistemdir ve Google'ın Dremel (Melnik et al., 2010) mimarisinden esinlenilmiştir. Drill, Calcite ve BigDAWG araçlarının hepsi birleşim işlemlerini tablo biçimi üzerinden yapmaktadır.

(Dasgupta et al., 2016) sosyal veri çözümlemesini de desteklemek için AWESOME isimli bir çoklu depo tabanlı sistem geliştirmişlerdir. AWESOME sistemi çözümlemeli uygulamalarda kullanılan türetilmiş veriyi işleyebilmek için tasarlanmıştır. ADIL adında tasarlanılan veri alınım dili (a data ingestion language) ile ham ve türetilmiş veri üzerinde taşıma ve çözümleme

[10] https://calcite.apache.org

[11] https://drill.apache.org

yapılabilmektedir. Ayrıca çalışmada veri yerleştirme ile ilgili birçok eniyileştirme stratejisi ortaya konmuştur. (Zheng et al., 2021, 2023) yaptıkları çalışmalarda PoliSci veri kümesini kullanarak PostgreSQL, Neo4j ve Solr ile üç depolu bir çoklu depo kurulumu yapmışlardır. PoliSci kapsamında 5 farklı veri kümesini bu depolara yüklemişler ve AWESOME ile maliyete göre depolar arası dinamik olarak veri dönüşümü sağlayarak kapsamlı bir değerlendirme yapmışlardır. ADIL betikleri ile üretilen sorgularda; farklı çoklu depo maliyet planları birbirleri ile ve tekil depolar ile kıyaslanmış olup aracın etkinliği ön plana koyulmuştur. Ayrıca AWESOME aracı halihazırdaki çoklu depo sistemlerinin önemli teknik özellikleri bakımından bu sistemler ile kıyaslanmıştır.

(Podkorytov et al., 2017) çoklu depo sistemlerini kullanacak olan uç kullanıcıların karmaşık ve detaylı sorgular ile ilgilenmediğine ve daha çok metin tabanlı arama ve çözümleme yapma ihtiyaçlarına dikkat çekmişlerdir. Buna göre (Gubanov, 2017)'un önerdiği POLYFUSE isimli mimari üzerinde çalışacak karmaşık çözümlemeler gerektiren sorgular tanımlanmış ve güçlükler ortaya konmuştur. HYBRID.POLY adında, kullanıcıyla etkileşime izin veren, bellek-içi (in-memory) ve çözümlemeli bir çoklu depo sistemi önerilmiş ve mimari detayları belirtilmiştir.

(Du et al., 2017) bir veri ambarında farklı tipte veri saklama motorlarının yer alabileceği durumu ele almışlardır. Buna göre çoklu depolarda veri alınım (ingestion) yapılabilmesi için dinamik ve iş-yükü yönelimli bir veri yerleşim yaklaşımına dikkat çekmişlerdir. TPC-DI benchmark için S-Store ve Postgres'ten oluşan bir polystore durumu tanımlanıp sonuçlar ortaya konmuştur.

(Bonaque et al., 2016) geliştirdikleri TATOOINE isimli araçla heterojen veri kaynakları arasında bütünleştirme yapmayı amaçlamışlardır. Çoklu depo ortamına, karışık veri bireyi (mixed data instance) adını vermişlerdir. Örnek olarak kullandıkları karışık veri bireyinde gazeteler ile ilgili metin, ilişkisel ve RDF verilerini tutmuşlardır ve veri bütünleştirme için de RDF veri modelini seçmişlerdir.

(Leclercq and Savonnet, 2019) çalışmalarında çoklu depo sistemlerinde

oldukça yoğun olarak gerçekleşen veri dönüşümü işlemini kolaylaştıran bir yöntem ortaya atmışlardır. Yazarlara göre çoklu depo sistemleri genel olarak aşağıdan yukarı yönlü (bottom up), performans ve sorgu eniyileştirme odaklı tasarlanmaktadır. Bu durumda da mantıksal model ve fiziksel model mantığı uygun şekilde birbirinden ayrılmamakta ve esnekliği sağlayabilmeye odaklanılarak iyi bir şema tasarımı göz ardı edilmektedir. Tasarladıkları Tensör Veri Modeli (TDM: Tensor Data Model) altyapısında birleşmeli diziler (associative arrays) kullanılmaktadır. İlişkisel, sütun, çizge ve anahtar-değer veri depolarının modelleri anlatılmış ve bu modellerin TDM cinsinden dönüşümleri ortaya konmuştur. Dizi işlemlerini Spark aracı ve R dili üzerinde gerçekleştirmişlerdir. Tasarladıkları modeli gerçekleyerek sistemi 2017 Twitter Fransa seçim verisi üzerinde doğrulayıp sonuçları elde etmişlerdir. Modelin bu örnek kullanım durumu için gerçekleştirimi ve anlatımı da farklı bir çalışmada sunulmuştur (Leclercq and Savonnet, 2018).

(Karimov et al., 2019) çoklu depolar için PolyBench isimli bir benchmark tasarlamışlardır. Benchmark üç ana kullanım durumu ve iki test senaryosu üzerine kurulmuştur, ayrıca detaylı bir çözümleme yapabilmek için de bir ölçüt kümesi tanımlanmıştır. Çoklu depolar tekil depolar ile kıyaslanarak hangi durumlarda üstün geldiği ortaya konmaya çalışılmıştır. Farklı parametre ve kombinasyonlarla kapsamlı bir deneysel çözümleme yapılmıştır. Belirledikleri metriklerden çalışma zamanı (runtime); ana sorgunun çalışma süresini, bireysel çalışma zamanı (individual runtime); belirli bir depo üzerinde çalışacak alt sorgunun çalışma süresini, atıl süre (idle time); bir deponun hesaplama yapmadan boş olarak durduğu süreyi, yük (load); çoklu depo ortamının girdi verisi büyüklüğünü ifade etmektedir. BigDAWG ve Spark araçları, tekli ve çoklu depo kullanım durumları üzerinde, tek-atım (one shot) ve sürekli (continuous) sorgularda kıyaslanmış ve sonuçları ortaya konmuştur. Genel olarak çoklu depo sistemlerindeki ana darboğazın veri transferi olacağı öngörülmektedir. Bu çalışma aynı zamanda çoklu depolar için ilk benchmark çalışması olma özelliğini de içermektedir. Benchmark'ın gelecekte yeni metrikler ve kullanım durumlarıyla daha kapsamlı olacak şekilde genişletileceği söylenmektedir.

(Agrawal et al., 2018) geliştirdikleri RHEEM (şimdiki adıyla Apache Wayang [12]) isimli araçla platformlar arası (cross-platform) bir veri işleme sistemi (data processing system) oluşturmuşlardır. Esinlenilen bakış açısında ise Hadoop veya bir DBMS gibi tek bir veri işleme çözümünün yeterli kalmadığı durumlarda uygun veri işleme platformunun seçilmesinin bir uzmanlık bilgisi gerektirdiğine vurgu yapılmıştır. Buna göre RHEEM aracı kendisine gelen bir görevin (task) çalıştırılacağı en iyi platformu seçmekle birlikte, gelen bu görevi alt görevlere bölerek her bir alt görevi belirli platformlar üzerinde eş zamanlı çalıştırarak genel maliyeti de düşürmektedir. Veri çözümleme görevlerini oluşturmak için bir arayüz, en iyi platformu seçebilmek için maliyet tabanlı özgün bir eniyileştirici ve farklı platformlar arasındaki görevleri yönetmek için bir çalıştırıcı olmak üzere üç ana bileşenden oluşmaktadır. Java, Scala, Python ve REST api desteklerini sunan araç Apache lisansına sahiptir ve oldukça olgunlaşmış bir yapıdadır.

(Kaitoua et al., 2019) çoklu depolar için Muses isimli dağıtık bir veri taşıma sistemi tasarlamışlardır. Hadoop, Spark, Flink, Postgres ve SciDB veri depolarını desteklemekle birlikte bunların arasında veri dönüştürüp taşıyarak ardışık düzenin (pipeline) performansını %30 kadar artırabilmektedir. Değerlendirmede çoklu depo performansı tekil depolar ile kıyaslanmakla birlikte, dönüşüm performansı da Akka kullanılarak ve ayrıca soket mimarisi üzerinden birbiriyle kıyaslanmıştır. Ortaya konan sonuçlar ile Muses'ın çoklu depo sistemlerinin performansını artırdığı ispatlanmıştır.

(Khan et al., 2019) anlamsal web ve çoklu depo sistemlerini birleştirerek PolyWeb isimli hibrit bir web çoklu deposu yaratmışlardır. Sağlık alanında yer alan, farklı tipte veriler içeren ve çok büyük boyutlara ulaşabilen kanser genom bilimi (cancer genomics) veri kümelerini sorgulamaya uygun doğal (native) veri yapılarına göre bölümlendirmekte, bunları literatürde varolan R2RML ve RML gibi eşleme yaklaşımları kullanarak RDF'e eşleyerek kaynak seçimi (source selection) yapmakta ve daha sonra birleştirilmiş (federated) Sparql sorgusu işler gibi işlemektedir. Çoklu depo ilişkisel, CSV ve RDF veri

[12]https://wayang.apache.org

yapıları üzerinden yapılandırılarak dönüşümler sağlamaktadır. Ayrıca sorgu ve sorgu planı eniyileştirmeleri yaparak sisteme gelen sorguların daha hızlı çalıştırılmasını sağlamaktadır. Veri kümesi üzerinde yaratılan 10 adet farklı karakteristiklerdeki SPARQL sorgusu ile değerlendirme yapılmış ve PolyWeb çoklu deposu kendi içinde yalnızca RDf yapılandırması üzerinden FedX ve HiBISCuS araçları ile kıyaslanmıştır. Sonuçlara göre PolyWeb'in diğer sorgu motorlarına kıyasla çok daha etkin olduğu ortaya konmuştur.

(Vogt et al., 2018) bulut üzerinde dağıtık ve kendi kendine uyarlanabilir Polypheny-DB isimli bir çoklu depo sistemi tasarlamışlardır. Sistem kullanıcı gereksinimlerine göre global veri bölümleme (partitioning) ve çoğaltma (replication) yaparak dinamik olarak uyarlanabilir veri yönetimini sağlamaktadır. Bu iki özelliğin çoklu depo yaklaşımı ile birleştirilmesi sistemin özgün yanını oluşturmaktadır. (Vogt et al., 2021) önceki çalışmada kurmuş oldukları Polypheny-DB çoklu depo sistemini bir adım ileri götürerek hibrit işlemsel ve çözümlemeli işleme (hybrid transactional and anlytical processing - HTAP) sistemleri için bütünleştirme desteği eklemişlerdir. Böylece literatürde üzerinde fazla çalışılmamış bir alan olarak çoklu depo sistemlerinin OLTP ve OLAP işyüklerine uyumlaması sağlanmıştır.

Horizon 2020 araştırması altında desteklenmiş bir Avrupa Birliği projesi olan TYPHON[13]'un amacı, ölçeklenebilir hibrit veri sürekliliği mimarilerini tasarlamak, geliştirmek, sorgulamak, evrimleştirmek (evolution), çözümlemek ve izlemek için endüstri tarafından onaylanmış bir metodoloji ve bütünleşik bir teknik öneri sağlamak olarak tanımlanmıştır. Bu mimariler, projeyi otomotiv, yer gözlemi, bankacılık ve telekom alanlarından dört kullanım durumu üzerinden yönlendirerek Avrupa endüstrisinin artan ölçeklenebilirlik ve heterojenlik gereksinimlerini karşılamasını sağlamakta olduğu belirtilmiştir. Proje kapsamında hibrit çoklu depoların tasarlanması, model dönüşümleri, sorgulanması, veri erişimi ve işleme için bir çerçeve sunulması, çoklu depo içindeki verinin yönetimi ve gelişimi için bir metodoloji ve altyapı sağlanması gibi çok sayıda alt çalışma tanımlanmıştır. Bu kapsamda (Basciani et al.,

[13]https://www.typhon-project.org

2020) TyphonML isimli bir modelleme dili ortaya koyarak kullanıcıların hangi verinin hangi depoda sorgulanacağını belirtmesine olanak tanımaktadır. (Fink et al., 2020) çoklu depodaki sorgu değişimleri için TyphonML ve TyphonQL kullanılarak dinamik sorgu uyarlaması üzerine bir çalışma ortaya koymuşlardır. (Kolovos et al., 2019) hibrit çoklu depoların modellenmesi, dağıtımı (deployment) ve sorgulanması için TYPHON kapsamında geliştirilen TyphonML, TyphonDL ve TyphonQL dillerini ve bunların birlikte çalıştığı mimariyi tanıtmışlardır. (Benats et al., 2022) hibrit çoklu depoların sürekli evrimi üzerine bir çalışma ortaya koymuşlardır. Çoklu deponun TyphonML ile tanımlanan veri ve şema eşlemelerini, bu yapılandırma üzerindeki sorgu işletim performanslarını ve her bir deponun zaman içerisinde değişen veri boyutunu baz alarak yeni bir çoklu depo şema yapılandırması önermektedir. Buna ek olarak gerekli veri dönüşümlerini gerçekleştirerek yeni indeksleri oluşturmaktadır.

(Ben Hamadou et al., 2019) gelişen veri bilimi disiplini ile ortaya çıkan veri çözümlemesi gereksinimini geliştirmiş oldukları veri uzayı (dataspace) (Franklin et al., 2005) tabanlı çoklu depo mimarisi ile karşılamışlardır. İlişkisel, belge ve sütun heterojen veri modellerinin yönetimini en üstte veri uzayı katmanı vasıtasıyla ele almışlardır. OLAP uygulamalarında en çok kullanılan GPSJ (generalized projection, selection and join) sorguları desteklenmiştir. Spark aracının üzerine kurulan sistem, çapraz veritabanı sorguları birleşimli ilişkisel cebir (nested relational algebra) veya diğer adıyla iç içe döngülü birleşim ile işlenmektedir.

(Poudel et al., 2019); görüntü, metin, anahtar-değer ve çizge gibi çok farklı tipte verilerden oluşan ve sürekli gelişen astronomi alanındaki heterojen yapıyı yönetebilmek için bir çoklu depo mimarisi ortaya atmışlardır. PTF, iPTF ve ZTF veri kümeleri kullanılarak bir veri bölümlendirme ve işleme yapısı tasarlanmıştır. Mimari detayları ortaya konan sistem üzerinde ilerletilen çalışmada (Poudel et al., 2022) ZTF (Zwick Transient Facility) verisi üzerinde çoklu depo yapılandırması için ortaya atılan veri modellerinin dönüşümlerine ve bütünleştirilmesine çalışılmıştır. SQL sorguları üzerinden sorgulama yapılarak

depolara özel sorgu dönüşümleri arka planda yapılmaktadır. Veri dönüşümü ve bütünleştirmesi ise bağlı veri üzerinden RDF veri modeli ile gerçekleştirilmiştir. Sistemin değerlendirilmesi için astronomi verisi üzerinde tekil ve bileşik toplam 20 sorgu oluşturulmuştur. Ölçeklenebilir olarak tasarlanan sitemin etkinliği elde edilen sonuçlara göre ortaya konmuştur.

(Singhal et al., 2019) heterojen çözümlemesel iş yükleri hızlandırılmış bir çoklu depo sistemi geliştirmişlerdir. Varolan çoklu depo sistemlerinin donanım hızlandırıcıları (GPU, FPGA, CGRA vb.) kullanılarak performansının artırılmasına yönelik bir mimari ortaya atılmıştır. Sistemin mimari detaylarının yanında donanım hızlandırıcıları kullanılırken farklı veri modelleri arasında dönüşümlerin nasıl yapılacağı anlatılmıştır. Çözümlemesel sistemler hedeflenmiş olup, yoğun iş yükü gerektirmesinden dolayı bilindik CPU yöntemiyle başa çıkılamayacak olduğu ortaya atılmıştır.

Cafarella et al. (2021) çoklu depo sistemlerinin kullanımına farklı bir bakış açısı getirerek DBOS isimli çoklu depo tabanlı bir veritabanı işletim sistemi geliştirmişlerdir. Varolan işletim sistemlerinin günümüz hesaplama ortamlarının (computing environment) çok öncesinde tasarlanmış olduğundan ölçeklenebilirlik, heterojenlik, uygunluk ve gizlilik gibi zorluklarla başa çıkmasının zor olduğuna vurgu yapılmıştır. Yenilikçi bir işletim sistemi olarak veri-merkezli bir mimari ortaya atılmış; buna göre işletim sisteminin tüm bileşenleri veri depoları üzerine dağıtılmakta ve işletim sisteminin üzerindeki tüm işlemler ise sorgular ile tanımlanmaktadır. Böylece işletim sisteminin kolay bir şekilde ölçeklenebilmesinin önünün açılmış olacağı ileri sürülmektedir. Önerilen yakalşımın olası tüm faydaları tartışılmış ve tasarlanan mimari model anlatılmıştır.

Doktora öğrenimi içerisinde varlık bağlama yaklaşımı (entity linking) ile çoklu depo sistemlerinin bütünleştirildiği bir çalışma (İnan et al., 2017, 2018) ortaya konmuştur. Buna göre belirlenen varlık bağlama yöntemiyle yapısal olmayan metin verilerinden varlık bilgileri saptanmakta ve çoklu depo ortamında saklanmaktadır. Saptanan tanımlı varlıkların çizge deposunda, tanımlı varlıklara ait belgelerin, belge deposunda, bu belgelerdeki

kelime sıklıklarının ve kelimelerin geçtiği metinlerin ise anahtar-değer veri-tabanında saklanması önerilmiştir. Kullanım durumu tasarımı için sinema alanı seçilmiştir. Kurgulanacak çoklu depo sisteminde verilerin çeşitli ve farklı modellerdeki veri depolarında bütünleşik olarak saklanması ve sorgulanması planlanmış, böylece hızlı sorgulama ile veri beslenerek varlık bağlama ardışık düzeninin etkinliğinin artırılması amaçlanmıştır. Bu çalışma alan bağımlı bir çoklu depo sisteminin tasarlanması ve farklı bir sistem ile birlikte ardışık düzen içinde bütünleşik çalışmasının planlanması için bir ön çalışma niteliği taşımış olup tez sürecindeki kapsamlı sistemin geliştirilmesine fayda sağlamıştır.

4 YÖNTEM

Bu bölümde tezde ortaya konan özgün değer ile birlikte önerilen çoklu depo sistemi mimarisi detaylandırılmaktadır.

4.1 Veri Kümesi Seçimi ve Analizi

Bu bölümde tez kapsamında geliştirilecek olan çoklu depo sisteminin üzerinde çalışacağı veri ortamını oluşturmak amacıyla seçilen veri kümesi anlatılacaktır. Tıp alanından seçilen veri kümesine erişebilmek için öncelikle CITI Programı[14] (The Collaborative Institutional Training Initiative) altında sunulan tıp etiği kursunu ("Data and Specimens Only Research") tamamlayıp sertifikalandırmak gerekmektedir. Dolayısıyla öncelikle bu kurs tamamlanmış, tamamlama raporu alınarak PhysioNet üzerinden sunulan veri kümesine erişim izni istenmiş ve sonucunda da izin alınarak veri kümesi elde edilmiştir. İlerleyen bölümlerde seçilen bu veri kümesi detaylandırılarak açıklanacaktır.

4.1.1 MIMIC-III Veri Kümesinin Literatürdeki Önemi

MIMIC-III (Johnson et al., 2016); sağlık alanıyla ilgili ve çok sayıda hastayla bağlantılı verileri barındıran serbestçe erişilebilir (freely accessible) büyük bir veri kümesidir ve "Yoğun Bakım için Medikal Bilgi Pazarı" (Medical Information Mart for Intensive Care) anlamına gelmektedir. ABD'nin Boston şehrinde yer alan BETH Israel Deaconess Medical Center hastanesinde 2001 - 2012 yılları arasında yoğun bakımda yatan 40000'den fazla kişinin verileri toplanmıştır. Veriler, kimliksizleştirme (deidentification) yöntemi kullanılarak tıp etiğine uygun olarak korunması sağlanmıştır. Veri kümesi; hasta popülasyonu ile ilgili istatistiksel demografik bilgiler, yatakta saatlik olarak ölçülen yaşamsal belirtiler, laboratuvar test sonuçları, tedavi yöntemleri, ilaç tedavileri, hasta bakıcı notları, görüntüleme raporları (örneğin; radyoloji bölümünden), ölüm oranı vb. birçok bilgi içermektedir. Proje BETH hastanesinin kurumsal dene-

[14] https://about.citiprogram.org

tim kurulu (IRB: Institutional Review Board) tarafından ve MIT Üniversitesi tarafından onaylanmıştır. Veri kümesi PhysioNet (Goldberger et al., 2000) üzerinde açılmış, erişim ve kullanımı için gerekli adımlar, veri kümesinin web sitesindeki dökümantasyonda tanımlanmıştır. MIMIC-III birçok araştırma çalışmasına olanak sunmaktadır. Bunların içinde; hasta sonuçları için makine öğrenmesi yaklaşımları, tansiyon izleme tekniklerinin klinik etkileri, yapısal olmayan hasta notlarının çözümlemesi gibi birçok alan sayılabilir. Ek olarak MIMIC'in veri çeşitliliğinin yüksek olması dolayısıyla veri bütünleştirme araştırma alanında kullanımı da oldukça uygundur. Bu bağlamda veri kümesinin eski versiyonu olan MIMIC-II, MIT üniversitesi büyük veri çalışma grubu tarafından BigDAWG (Kolev et al., 2016d; Bondiombouy et al., 2016; Bondiombouy and Valduriez, 2016) çoklu depo sisteminin veri ortamını oluşturmak için de kullanılmıştır. İlerleyen kısımda MIMIC-III veritabanını oluşturan tablolar incelenerek içerdiği veriler ve kullanımı hakkında bilgi verilecektir. MIMIC-III veri kümesinin yeni versiyonu MIMIC-IV (Johnson et al., 2020, 2023) yayınlanmış olup, bilgi deposu hâlâ geliştirim aşamasındadır ve metin bilgileri gibi önemli bilgilerin yayınlanması henüz tamamlanmamıştır. Bu sebeple MIMIC-IV'ün kullanımı henüz yaygınlaşmamış olup, MIMIC-III oldukça aktif ve yoğun olarak kullanılmaya devam etmektedir.

4.1.2 MIMIC-III Veri Modeli Analizi

Klinik alan karmaşık yapısıyla birçok farklı bileşenden oluşmaktadır ve bu bileşenlerin her biri de birbirinden farklı gereksinimlere sahip olabilmektedir. Buna göre değerlendirildiğinde MIMIC-III veri kümesi klinik alandaki yoğun bakım birimiyle bağlantılı ve farklı yapılardaki hasta kayıtları ile ilgili genel bilgileri, hasta yatışları, laboratuvar analizleri, hastanın yatakta izlenen sürekli verileri, doktor notları, tedaviler, reçeteler, hastanın farklı birimler arası taşınma bilgileri vb. çok çeşitli verilerden oluşmaktadır. MIMIC-III verisi ilişkisel modele göre modellenmiştir ve içerisinde yer alan tüm veriler, birçoğu ilişkisel yapıya sahip farklı veritabanlarına kolaylıkla ve doğrudan yüklenebilir durumdadır. Üstün yöntemler (best practices) kullanılarak alan odaklı

gereksinim analizi yapılmış olup bu kapsamda çıkarılan veri tüketimi (data consumption) gereksinimlerine göre bir veri modelleme kurgusu önerilmiştir. Buna göre veriler tablo bazında incelenmiş ve 26 tablonun analizi sonucu belirlenen; ilişkisel, zaman serisi, anahtar-değer ve arama & belge odaklı 4 ana veri deposu gereksinimi ortaya konmuştur. Her durumda kesin olmamakla birlikte MIMIC-III veri kümesini referans alan çalışmalardaki kullanım durumlarında tasarımın kolaylaştırılması için verinin genel bir veri modeli çerçevesini çizmek amaçlanmıştır. İlerleyen alt bölümlerde belirlenen her veri deposu için kapsanan tablolara, bu tablolarda içerilen verilere ve her bir tablo için seçilen veri modelinin uygunluğuna açıklık getirilecektir.

4.1.2.1 İlişkisel Veriler

İlişkisel veri modeli gereksiniminde; yapısal verilerin tutulduğu, güncellemenin çok yoğun yapılmadığı, dolayısıyla da nispeten daha az miktarda veri barındırılan, veri bütünlüğünün önem arz ettiği durumlar ön plana çıkmaktadır. Buna göre Admissions, Patients, ICUStays, Transfers, Services, Callout, MicrobiologyEvents, Prescriptions, D_CPT tabloları ilişkisel yapıda tutulmaya uygun durmaktadır. Belirlenen tüm bu tablolar ve içerdikleri veriler aşağıda sırasıyla açıklanmaktadır.

- **Admissions:** Her hasta için belirli tarihler arasındaki hastaneye kayıt kabulü ve yatışı ile ilgili genel bilgileri bulunur. Hastanın hastanede geçirdiği sürenin takibi amaçlanır. Bu tablo neredeyse diğer tüm tablolar ile bağlantılıdır ve önemli bilgiler barındırmaktadır. Dolayısıyla bu sebeple ve sıklıkla güncellenmiyor olmasından ötürü ilişkisel yapıda tutulması uygun olacaktır.
- **Patients:** Hasta ile ilgili genellikle sabit kişisel bilgiler tutulur. Hastanın ölüm bilgisi olmadığı durumda herhangi bir güncelleme yapılmaz. Dolayısıyla bu veri parçası için ilişkisel veri modeli kullanılacaktır.
- **Transfers:** Yoğun bakım biriminde olsun veya olmasın hastanın hastane içindeki tüm hareketlerini içeren verileri (kayıtları) barındırır. ICUStays

tablosunda ise yalnızca yoğun bakım birimindeki kaldığı süredeki bilgiler bulunmaktadır. Bakım birimleri (care unit) WardID (Koğuş id)'lere göre tutulmaktadır. Bir koğuş belirli bir bakım birimine ait olabilmektedir ve bir bakım birimi içinde de birden fazla koğuş bulunabilmektedir. Ancak veri 12 yıllık eski bir veri olduğundan eskiden bakım birimlerine bağlı olan koğuşlar yeni bir kayıtta bir başka birime bağlı olabilecektir. Güncellemenin ve dolayısıyla da kayıt sayısının az olması bütünlüğü de korumak açısından verinin ilişkisel modelde tutulmasını gerektirmektedir.

- **ICUStays:** Transfers tablosundan türetilerek oluşturulmuş bir tablodur. Transfers tablosunu ICUStay_ID üzerinden gruplamakta, ICUStay_ID olmayan satırları elemektedir ve bir nevi filtreleme görevi görmektedir. Dolayısıyla veri yapısı Transfers tablosuyla aynı olup ilişkisel model kullanılacaktır.
- **Services:** Hastaların kabul edilip bakıldığı klinik servis bilgilerini içermektedir. Bu tablo aracılığıyla hastanın bakıldığı servisler veya bir serviste bakılan hastalar gibi bilgileri sorgulayabilmek mümkün olmaktadır. Hastanın bakıldığı yerin (lokasyon) bilgisini barındırır, tedavi bilgisi ise içermemektedir. Temel olarak önceki ve şimdiki servis bilgileri ve transfer tarihi bilgisini barındırmaktadır. İçerdiği kayıt sayısı, veri karmaşıklığı ve güncelleme sıklığının düşük olması ve ilişkisel yapıdaki diğer verilerle bağlantısı sebebiyle ilişkisel modelde tutulması uygun olacaktır.
- **Callout:** Hastaların yoğun bakım birimlerinden tahliye kayıtlarının tutulduğu tablodur. Hastanın yatış ve tahliye işlemi ile ilgili prosedürel veriler içerir. Genellikle hastanın hastaneden ayrılacağı, iyileştiği veya kötüleştiği zaman gereksinim duyulan bir işlem olduğundan az güncelleme olmakta ve yüksek bütünlük gerektirmektedir. Bu nedenle de ilişkisel model veri için oldukça uygundur.
- **Prescriptions:** Hastaya verilen ilaçlarla ilgili tedavi bilgileri (dozu, adı, birimi, tipi) yer almaktadır. İlgili ilaçların ne kadar süre ile ve hangi tarih aralığında verildiği bilgilerini içermektedir. Nispeten büyük boyutlu (4

milyon kayıt) bir veri olsa da tabloya kayıt periyodu zaman serisinde tutulacak verilere kıyasla düşüktür. Bu yüzden ilişkisel yapıda tutulması daha uygun olacaktır.

- **D_CPT:** CPT (Current Procedural Terminology) kodlarıyla ilgili bölüm (section) ve alt bölüm (subsection) aralıkları bilgilerini ve açıklamalarını barındırmaktadır. Normalde başında D harfi ile başlayan tablolar sözlük (dictionary) bilgisi tutmakta ve anahtar değer veri yapısına daha çok uymaktadır. Ancak burada aralık değerleri tutulduğu için ilişkisel veri yapısını kullanmak daha mantıklı olacaktır.
- **CPTEvents:** Bu tablo hastalara uygulanan tedavi işlemi kodlarının (CPT - Current Procedural Terminology codes) hastalarla eşlemesini tutmaktadır. Buna göre hastaya uygulanan tedaviler ve müdahaleler belirlenmiş olup ileride faturalama aşamasında kolaylık sağlamaktadır. Tedaviyi uygulayan maliyet merkezi (cost center), tedavinin uygulandığı hasta, uygulanan tedavinin CPT kodu, kodun kategorisi ve açıklaması gibi bilgiler yer almaktadır. Tablo 573146 kayıttan oluşmaktadır. Faturalamaya yardımcı olma amaçlı bir tablo olup yoğun sorgu yükü altına girmemektedir. Dolayısıyla ilişkisel modelde tutulması daha uygun olacaktır.
- **DRGCodes:** Hastalarla ilgili tanılarla ilişkili grupların kodlarını (DRG - Diagnoses Related Groups codes) içermektedir. Tablo; hastayla ilgili DRG tipi, kodu, önemi, tanım, ölüm oranı gibi bilgileri barındırmakta ve 125557 kayıttan oluşmaktadır. Tablo anahtar-değer modeline uygun yapıda olmadığından ilişkisel veri modelinde yer alması planlanmıştır.

4.1.2.2 Zaman Serisi Veriler

Zaman serisi veriler basit tanımıyla zaman içerisinde izlenen, gözlemlenen ve toplanan ölçümler veya olaylardır. Zaman serisi veritabanları (time-series databases) özellikle zaman damgalı metrikleri, olayları veya ölçümleri işlemek için tasarlanmış olup zaman içindeki değişimi ölçmek için eniyileştirilmiştir.

Yaşam döngüsü yönetimi (lifecycle management), özetleme (summarization) ve çok geniş tarih aralıklarında kayıt tarama özellikleriyle diğer veritabanlarından farklılaşmaktadır. MIMIC-III'te yer alan tablolardan isminde "event" geçenlerin çoğu zaman içerisinde toplanan yoğun miktarda verilerden oluşmaktadır. Dolayısıyla bu tablolar zaman serisi veri modeline göre saklanmaya adaydır. MIMIC-III veri kümesinde zaman damgası (timestamp) içeren tüm bilgiler her bir hasta için kendi içinde tutarlıdır ve buna göre kimliksizleştirilmiştir (deidentified). Dolayısıyla belirli tarih aralığındaki birden fazla hasta bilgisinin sorgulanması mümkün gözükmemekte ancak bir hastanın belirli tarih aralığındaki bilgileri sorgulanabilmektedir. Bu sorun, zaman serisi veritabanında saklanacak tablolardaki tüm tarihlerdeki yıl değerlerinin min-max normalleştirme (normalization) yöntemiyle düzenlenmesi sonucu giderilmiştir. Buna göre chartevents, datetimeevents, inputevents, outputevents, procedureevents_mv, labevents tabloları zaman serisi veri yapısına göre saklanmaya uygundur. Şimdi bu tabloları ve içerdiği verileri sırasıyla açıklayalım:

- **Chartevents:** Hastanın yoğun bakım birimindeki sürekli izlenen gözlem bilgileridir. Elektronik çizelge (electronic chart) hastanın rutin yaşamsal değerlerini barındırır. En yoğun miktarda veri kaydı (tuple) bulunan tablodur ve hastanın yatışı boyunca toplanan zaman damgalı periyodik verilerini içermektedir, dolayısıyla da zaman serisi veri modeline oldukça uygundur.
- **Datetimeevents:** Yoğun bakım hastalarıyla ilgili daha uzun süreli periyoda sahip tarihsel ölçümlerin (son diyaliz tarihi gibi) olduğu verileri içerir. Chartevent tablosundaki rutin ölçümler dışındaki tüm tarihsel ölçümler tutulur ve yapı olarak da benzerlik gösterdiğinden zaman serisi veri modelinde saklanmaya uygundur.
- **Inputevents:** Hastaya verilen sıvı (serum, ilaç vb.) bilgileri tutulmaktadır. Bunlar zaman damgalı ve periyodik olduğundan zaman serisi modeline göre saklanacaktır.
- **Outputevents:** Hastadan alınan sıvı (kan, idrar, tükürük) bilgileri

bulunmaktadır. Bunlar da zaman damgalı ve periyodik olduğundan zaman serisi modeline göre saklanacaktır.

- **Procedureevents:** Hastaya uygulanan tedavi bilgileri yer almaktadır. Zaman bağımlı olarak değişiklik gösterdiğinden, zaman serisi veri yapısında tutulacaktır.
- **Labevents:** Hastalar ile ilgili tüm laboratuvar tabanlı ölçümleri içermektedir. Zaman damgalı ve hasta ile ilişkilendirilmiş belirli bir metrik ve bunun ölçüm değerini içerir.
- **MicrobiologyEvents:** Hastaların zaman damgalı mikrobiyoloji ölçüm ve duyarlılıklarını içerir. Kan, idrar, tükürük gibi örnekler içinde mikroorganizma ile ilgili ölçüm ve direnç vb. bilgiler bulunur.

4.1.2.3 Anahtar Değer Veriler

Tanımlayıcının (identifier) belirli olduğu ve bu tanımlayıcıya göre de indeksli basit yapıda verilerin hızlı ve kolayca erişildiği veri yapısıdır. Anahtar-değer eşlileri bir koleksiyon olarak bunun için özelleşen veritabanında saklanır. Anahtar, eşsiz tanımlayıcı (unique identifier) görevi görerek bilginin veritabanında etkin saklanmasını ve hızlı erişimini sağlar. MIMIC-III veri kümesinde sözlük olarak (dictionary) tutulan tabloların neredeyse hepsi anahtar-değer veri yapısında tutulmaya uygundur. Buna göre Caregivers, Diagnoses_ICD, Procedures_ICD, D_ICD_Diagnoses, D_ICD_Procedures, D_Items, D_LabItems tabloları bu yapıya uygun olup sırayla açıklanacaktır:

- **D_Items:** Veritabanındaki kalem bilgileri (açıklama ve kısa bilgi) yer alır. ItemId $\rightarrow$ [Label, Abbreviation, DBsource, Linksto, Category, Unitname, Param_type, ConceptId] ve Label $\rightarrow$ [ItemId] yapısıyla anahtar-değer olarak tutulmaya uygundur.
- **D_LabItems:** Laboratuvardaki kalem (açıklama ve kısa bilgi) yer almaktadır. ItemId $\rightarrow$ [Label, Fluid, Category, Loinc_code] yapısıyla anahtar-değer olarak tutulmaya uygundur.

- **D_ICD_Diagnoses:** ICD kodu ve tanı açıklaması bilgisini içerir. ICD9_Code → [Short_Title,Long_Title] yapısıyla anahtar-değer olarak tutulmaya uygundur.
- **D_ICD_Procedures:** ICD kodu ve tedavi açıklaması bilgisini içerir. ICD9_Code → [Short_Title,Long_Title] yapısıyla anahtar-değer olarak tutulmaya uygundur.
- **Caregivers:** Hastabakıcıların kategorilendirildiği tablodur. Hastabakıcının görev kodu ve açıklaması bulunur. CGID → [Label, Description] yapısıyla anahtar-değer olarak tutulmaya uygundur.
- **Procedures_ICD:** Hasta → Uygulanan Tedavi eşlemeleri tutulur ve Subject_id-Hadm_id → Seq_num:ICD9_Code anahtar-değer olarak tutulmaya uygundur.
- **Diagnoses_ICD:** ICD (International Classification of Diseases) sistemiyle kodlanıp kaydedilen hasta tanıları tutulmaktadır. Subject_id-Hadm_id → Seq_num:ICD9_Code yapısıyla anahtar-değer olarak tutulmaya uygundur.

4.1.2.4 Arama ve Belge Odaklı Veriler

Yapısal halde olmayan metin tipindeki işlenmemiş veriden bilgi elde edileceği zaman metin içinde arama yapmak gerekecektir. Metin verisinin yanında, bu veri ile bağlantılı ve ilgili metni tanımlayan ve detaylandıran yapısal birtakım verilerin de tutulması icap edebilecektir. Klasik ilişkisel veri modeli yapısal ve tanımlayıcı verilerin tutulmasına özelleşmiştir. İlişkisel model metinsel aramaya özel yapılar içermediğinden büyük metin belgelerinin burada saklanması ciddi bir darboğaz (bottleneck) oluşturacaktır. Arama motorları, yapısal olmayan metinsel veriler üzerinde arama yapmaya özelleşmiş çözümlerdir. Metinler kendilerini tanımlayan yardımcı yapısal verilerle birlikte belge (document) tipinde ve genellikle JSON veya XML formatlarında tutulmaktadır. MIMIC-III'te hastalar ile ilgili çok sayıda metinsel not verisi yer almakta ve bunlar NoteEvents tablosunda tutulmaktadır.

- **NoteEvents:** Tabloda kimliksizleştirilmiş (de-identified) hasta notları yer almaktadır ve bunlar; hemşire ve doktor notları, ECG raporları, taburcu özetleri gibi büyük boyutlu yapısal olmayan metin bilgileridir. Bunun yanında tabloda notla ilgili tarih, kategori vb. yapısal bilgiler de yer almaktadır. Hem metinsel arama hem de yapısallık söz konusu olduğunda belge tabanlı arama motorları en uygun ve kaçınılmaz çözüm olarak karşımıza çıkmaktadır.

Anlatılan bu dört tipteki veri modeli için kullanılacak veritabanı teknolojileri belirlenmiştir. Buna göre ilişkisel veri modeli için → PostgreSQL, zaman serisi veri modeli için → InfluxDB, anahtar-değer veri modeli için → Redis ve arama & belge odaklı veri modeli için de → Elasticsearch kullanılmıştır.

4.1.3 MIMIC-III Veri Kümesi Kullanım Çözümlemesi

Bu bölümde veri kümesinin kullanım alanlarının ve kapsamının belirlenmesine açıklık getirilecektir. Buna göre öncelikle MIMIC-III'e referans veren çalışmaların veri kümesinin hangi parçalarını, hangi alanlarda ve hangi amaçlarla kullandığına değinilecektir.

Günümüzde hızla yaygınlaşıp gelişen makine öğrenmesi ve derin öğrenme uygulamaları büyük miktarlarda, iyi tanımlanmış ve güvenilir veriye ihtiyaç duymaktadır. Bu verinin kullanım durumunun faydalı ve kullanışlı olması da yapılan çalışmanın değerini artırmaktadır. MIMIC-III veri kümesi klinik tıp alanında 12 yıl boyunca yoğun bakım biriminin hastane bilgi sistemlerinden toplanan ve bütünleştirilip düzenlenen çok çeşitli verilerden oluşmaktadır. Sağlık alanının başlı başına önemli olması sebebiyle, bu alanda verilecek katkının da kıymetli olacağı açıktır. Dolayısıyla veri bilimi alanında artan çalışmalar arz ettiği önem sebebiyle de MIMIC-III veri kümesini kullanmaya ve sağlık alanında çok disiplinli katkı vermeye başlamışlardır. 2019 Aralık ayından bu yana MIMIC-III'e referans veren çalışmaların sayısı 1100'den 5700'e yükselmiştir. Sadece bu durum bile sağlık alanındaki bu verinin önemini ortaya koymaya yetmektedir. MIMIC-III veri kümesine referans veren çalışmalar önemli ölçüde taranmış olup, bu çalışmalardan veri kümesi içinde en

çok çalışılan kullanım durumları belirlenmiştir. Buna göre kullanım durumları; sepsis (kan zehirlenmesi) teşhisi, ölüm oranı tahmini, yapısal olmayan veri içinde ICD-9 (international classification of diseases) kodlarının tahminlenmesi, kalp krizi tahminleme, klinik metinlerden anlamlı bilgi çıkarımı, mekanik ventilasyon optimizasyonu, sıralı tanı tahminleme, veri bütünleştirme, hasta notlarının kimliksizleştirilmesi, hastaların yoğun bakım birimine planlanmamış geri kabulü, hastaların yatış sürelerinin tahminlenmesi vb. gibi çok sayıda senaryodan oluşmaktadır. Bu senaryolardan önemli olanlar ve bunların veri bütünleştirme bakış açısından değerlendirilmesinden Bölüm-4.1.5'teki Kullanım Senaryoları bölümünde bahsedilecektir.

4.1.4 Verinin İlişkisel Depoya Yüklenmesi

Belirlenen kullanım senaryoları veri bütünleştirme açısından kritik önem arz etmektedir. Bölüm-4.1.2'de genel hatlarıyla analiz edilen MIMIC-III veri kümesinin temelinde yer alması öngörülen veri modellerinden bahsedilmişti. Buna göre kullanım senaryolarının çoğunda, bir alt veri grubu, sorgu sonucu olarak elde edileceği zaman farklı depolarda tutulan verilerin birleşik (joined) olarak sorgulanması gerekli olacaktır. Bu durumda da verilerin çoklu depo ortamına uygun olarak tutulması ve sorgulanmasının sorgu performansını ve etkinliğini artırması öngörülmektedir. Dolayısıyla çoklu depo sorgularının kıyaslanacağı bir dayanak sıfır noktası oluşturmak amacıyla, MIMIC-III verisinin tamamı ilişkisel yapıda PostgreSQL veritabanına yüklenmiştir. Zaten MIMIC-III için tasarlanan genel veri modeli de ilişkisel olduğundan yükleme için gerekli tüm adımlar github üzerindeki MIMIC-III kod tabanında[15] mevcuttur. Buradaki adımlar takip edilerek veri kümesi yerel bilgisayar üzerine yüklenmiştir. Buradaki temel amaç senaryoların kapsadığı veriyi sunacak olan sorguların doğrudan ilişkisel model üzerinde çalıştırıldığında verdiği performansı sınamaktır. Buna göre ilişkisel modelde, belirlenen her senaryonun içerdiği sorgular çalıştırılarak sorgu çalışma süresi baz alınmış ve aynı veriyi

[15]https://github.com/MIT-LCP/mimic-code/tree/main/mimic-iii

getirecek sorgular tasarlanan çoklu depo ortamı üzerinde geliştirilen sorgu motoru aracılığıyla çalıştırılmıştır. Sonuç olarak çoklu depo sorgu performansı, ilişkisel model sorgu performansı ile kıyaslanarak tasarlanan çoklu depo ortamının ve sorgu motorunun gerekliliği ortaya konmuştur ve sonuçlar Bölüm-5.3'te detaylıca çözümlenmiştir.

4.1.5 MIMIC-III Kullanım Senaryoları

MIMIC-III veri kümesini kullanan çalışmalar, belirli bir alt kümeyi veya kümeleri içeren bir veya daha fazla senaryo üzerinde çalışmaktadırlar. Bu çalışmaların %98'lik bir kısmı makine öğrenmesi / derin öğrenme ile ilgilidir ve genellikle belirli benzer başlıklar altında toplanmışlardır. Bu başlıklar da kullanım senaryolarını oluşturmakta ve her biri belirli bir makine öğrenmesi konusunu ve/veya amacını ifade etmektedir. Makine öğrenmesinin dayandığı temel, veri bilimi olduğundan ilgili algoritmaların en temel yapı taşı da eğitim verisi olmaktadır. Dolayısıyla algoritmanın başarıya ulaşabilmesi için de burada eğitim verisinin özellik (feature) bakış açısından özenle seçilmesi gerekmektedir. Senaryodaki amaca ulaşmak için, gerekli verilerinin veri kümesinden hem etkin hem de hızlı çekilmesi makine öğrenmesi algoritmasını girdi olarak besleyip tahmin çıktısının hızlıca üretilmesi açısından önemlidir. Birbirlerinden çok farklı yapılarda olabilen bu veriler, çoklu depo ortamında tutularak sorgulandığında, tek bir depoda tutulmasına göre çok daha iyi bir sorgu performansına sahip olarak daha hızlı elde edilmesi beklenmektedir. Buna göre ilerleyen bölümde MIMIC-III ile bağlantılı kullanım senaryoları ile bu senaryolar için gerekli verilerden bahsedilerek bu verilerin veri kümesinde hangi tablolarda tutulduğu anlatılacaktır. Öncelikle MIMIC-III veri kümesine referans veren tüm literatür taranarak sağlık alanındaki gerçekçi kullanım durumları ortaya konmuştur. Bu kullanım durumlarından en çok çalışılan kritik senaryolar belirlenmiştir. Bu senaryoların MIMIC-III üzerinde ilişkili olduğu tablolar belirlenip çözümlenmiş ve oluşturulacak sorgularla ilişkili metrikler elde edilmiştir. Belirlenen kritik kullanım durumları ise sepsis tespit edilmesi, ölüm oranı tahminleme ve klinik metin sınıflandırmasıdır ve

her biri ilişkili olduğu önemli çalışmalar ve bağlantılı oldukları tablolardaki kritik özellikler (features) ile birlikte detaylandırılmıştır. Ortaya konan bu çalışma sağlık alanındaki önemli bilimsel çalışmaların gelişimine destek vermeyi amaçlaması ve gerçekçi olması dolayısıyla tezin en önemli katkılarından birisini oluşturmaktadır.

- **SEPSIS TESPİT EDİLMESİ:** Kan zehirlenmesi teşhisi (Raghu et al., 2017; Komorowski et al., 2018; Nemati et al., 2018; Wu et al., 2018; Desautels et al., 2016; Kong et al., 2020) hastanede yoğun bakım birimlerinde en sık rastlanan ölüm nedenlerinden birisi olması nedeniyle önem arz etmektedir.
  - EMR (electronic medical record) verisi, arterial blood pressure (arter kan basıncı), heart rate (kalp atış hızı) gibi veriler izlenerek tahminleme yapılması amaçlanmaktadır. (Chartevents, Inputevents, Outputevents, Labevents)
- **ÖLÜM ORANI TAHMİNLEME:** Hastane içi ölüm oranının (mortality prediction) bulunması (Johnson et al., 2017; Rajkomar et al., 2018; Che et al., 2018; Harutyunyan et al., 2019) tedavilerin ve müdahalelerin başarısı açısından önemlidir.
  - Zaman serisi veriler kullanılarak hastanın ilk 48 saatteki durumundan yararlanılarak yapılabilir. (Chartevents, Inputevents, Outputevents, Labevents)
  - ICU kayıt kabul (readmission'lara bakılarak) verisi temizlenip kullanılabilir. (Admissions)
  - SEPSIS olan hastalar için ölüm oranının tahminlenmesi
  - Sıvı alımı (fluid intake) ile ölüm oranı arasındaki ilişkinin bulunması (Inputevents, Chartevents)
  - Hemşire notlarındaki duygu durumu (sentiment) değerleri üzerinden hastane dışı ölüm oranı tahminlemesi (Noteevents)
- **KALP KRİZİ TAHMİNLEME:** Chartevents tablosu üzerinden ya-

şamsal değerlerin izlenerek kalp krizinin tahminlenmesi üzerine çalışılmaktadır. (Beaulieu-Jones et al., 2019)

  - Mean arterial blood pressure
  - Arterial systolic and diastolic blood pressures
  - Beats per minute
  - Respiration rate
  - Peripheral capillary oxygen saturation (spo2)
  - Mean noninvasive blood pressure
  - Mean systolic and diastolic blood pressures

- **KLİNİK METİN SINIFLANDIRMASI:** Hasta notlarının analiz edilerek hastalıkların, tanıların, tedavilerin vb. sınıflandırılması amaçlanmaktadır (Mullenbach et al., 2018; Prakash et al., 2017; Luo, 2017).
  - Noteevents tablosundaki klinik metinlerden faydalanılmaktadır.
- **FİZYOLOJİK DÜŞÜŞÜN TESPİT EDİLMESİ:** Hastalardaki fizyolojik düşüşün tespit edilmesi amaçlanmaktadır (Harutyunyan et al., 2019). (Patient, Admissions, ICUStays, Chartevents)
  - Organ yetmezliğinin tahminlenmesi: Hastanın yaşamsal değerleri 24 saat içinde aniden bozulacak mı vb.
- **MEKANİK HAVALANDIRMA OPTİMİZASYONU:** Yoğun bakımdaki hastalar için solunum desteği süresinin tahminlenerek, ne zaman kesileceğinin bulunması amaçlanmaktadır (Prasad et al., 2017). Hastanenin yoğun bakım biriminde, gerekli olsun veya olmasın hastaların çoğuna mekanik havalandırma desteğinin takılmasıyla bunun hastane içi maliyetleri artırdığına vurgu yapılmaktadır. Bu sebeple mekanik havalandırmanın optimize edilmesinin de ekonomik bakış açısıyla önemine değinilmiştir. Oksijen seviyelerinin ölçülerek mekanik havalandırmanın yalnızca belirli hastalara, doğru süre ile önerilmesi amaçlanmakta ve böylece maliyetler düşürülerek hastalar için en uygun tedavi yöntemi bulunmaya çalışılmaktadır.

  - Admissions, Patients → Weight, Age, Gender, Ethnicity
  - Chartevents → Arterial O2 pressure, Arterial PH, Inspired O2 Fraction, O2 Flow, Positive-end-expiratory pressure (PEEP) set, Plateau pressure, Mean Airway Pressure, Ventilator Mode
  - Inputevents → sıvı alımları (fluid intakes), saatlik izlenen metrikler, verilen ilaçlar vb.

- **YAŞAMSAL DEĞERLERİN İZLENMESİ:** Yoğun bakımdaki hastaların kritik yaşamsal değerlerinin izlenerek (vital sign monitoring) bu değerlerde bir düşüş ve/veya aşırılık tespit edildiğinde müdahale edilmesi amaçlanmaktadır. Burada anlık karar verme kritik olduğundan daha çok olay akış işleme (event stream processing) ve karmaşık olay işleme (complex event processing) vb. gibi yöntemler kullanılmaktadır. Yoğun bakım biriminde hastaların yataktan anlık olarak izlenen (bedside monitoring) verileri chartevents tablosu altında tutulmaktadır. Dolayısıyla zaman serisi veritabanı bu anlık verilerin doğru şekilde ele alınması için çok iyi bir çözüm olarak ön plana çıkmaktadır.
- **ÖNGÖRÜLMEYEN GERİ KABUL:** Bir hastanın taburcu edildikten sonra hastaneye tekrar başvurması (unplanned readmission) hem hasta açısından hem de hastane açısından önemli bir sorundur (Lu et al., 2019; Zhang et al., 2022). Öyle ki bu sorunu engellemek hem doğru tedaviyi ve hasta memnuniyetini sağlayacak, hem de hastanenin hasta bakımı ile ilgili maliyetlerini düşürüp prestijini yükseltecektir. Belirlenen ICD-9 kodları taburcu özetlerinin içinde aranarak bir karar vermeye çalışılmakta ve bu sayede öngörülmeyen geri kabulün önüne geçmek amaçlanmaktadır. Bu aşamada D_ICD_PROCEDURES, D_ICD_DIAGNOSES ve NOTEEVENTS tablolarından faydalanılmaktadır.
- **(UZATILMIŞ) KALIŞ SÜRESİ TAHMİNLEME:** Burada hastanın hastanede fazladan kalış süresini tahminlemek amaçlanmaktadır. Admissions tablosundan gerçek kalış süreleri elde edilebilmekte ve bu

kalış süresinin neden uzadığı farklı parametrelerle incelenmektedir.

- **OTOMATİK ICD-9 KODU ETİKETLENMESİ:** Notlardan hastalıkların belirlenip kodlar ile etiketlenmesi amaçlanmaktadır (Baumel et al., 2017; Mullenbach et al., 2018). D_ICD_PROCEDURES, D_ICD_DIAGNOSES, NOTEEVENTS tablolarından faydalanılarak taburcu özetlerindeki notlardan tanı (teşhis) belirlenmeye çalışılmaktadır.
- **FENOTİP SINIFLANDIRMASI:** Hasta (Patients), hastane başvuru (Admissions), yoğun bakım yatış (ICU stays) ve klinik çizelge olayları (Chartevents) verileri üzerinden hastaların fenotip (dış görünüş) sınıflandırması yapılmaktadır (Harutyunyan et al., 2019).
- **DİNAMİK TEDAVİ ÖNERİSİ:** Tanısı konan belirli hastalıklar için dinamik olarak tedavi önermek (dynamic treatment recommendation) amaçlanmaktadır (Wang et al., 2018). Prescriptions, CPTEvents ve Chartevents tablolarından ilaç tedavileri, hastalıklar ve fizyolojik parametreler gibi veriler kullanılmaktadır.
- **OLUMSUZ OLAYLARIN TAHMİN EDİLMESİ:** Yapısal elektronik sağlık kayıtlarından (Electronic Medical Record) olumsuz (adverse) olayların tahmin edilebilmesi için derin öğrenme vb. akıllı modellerin geliştirilmesi üzerine çalışılmaktadır (Tomašev et al., 2021).
- **KLİNİK METİNLERDEN BİLGİ ÇIKARIMI:** Noteevents tablosunda yer alan; doktor, hemşire, radyoloji, reçete, taburcu özeti, genel vb. şekilde sınıflandırılmış metinler doğal dil işleme yöntemleri ile çözümlenerek birçok kategoride bilgi çıkarımı yapılmaktadır. Bu alt alanlar ve son yıllarda yapılan ilgili çalışmalardan bazıları aşağıda listelenmiştir:
  - Bağlamsal kelime gömme modellerinin (contextual word embedding models) oluşturulması (Alsentzer et al., 2019)
  - Klinik kavram ve ilişki çıkarımı (Yang et al., 2020; Agrawal et al., 2022)
  - Eksik verilerin tamamlanması (Luo, 2022)

  - Bilgi çıkarımı ardışık düzeni (information extraction pipeline) geliştirimi (Wang et al., 2020; Zhang et al., 2021)
  - Notlardan yaşamsal değerlerin (vital signs) çıkarımı (Khurshid et al., 2022)
  - Etiketlerden faydalanarak medikal görüntü tanıma (Huang et al., 2021)

- **MEDİKAL KAYITLARIN BİLGİ ÇİZGELERİNE BİRLEŞİMİ:** MIMIC'teki medikal kayıtların üst düzey ontolojiler ile birleştirilmesi ve temsil edilen bilginin güçlendirilmesi amaçlanmaktadır (Wang et al., 2017b). Varlık bağlama (entity linking) kullanarak MIMIC-III verisinin RDF modeline çevrilerek Drugbunk ve ICD-9 ontolojileri ile bağlantısını sağlayan bir çerçeve önerilmektedir.
- **ÇOKLU DEPO SİSTEMLERİ:** Son olarak MIMIC (II ve III) veri kümesi sunduğu veri çeşitliliği sebebiyle çoklu depo sistemlerinin tasarlanması ve işlevselliğinin ortaya konması açısından çok önemli bir veri kaynağıdır (Duggan et al., 2015; Khan et al., 2019).

4.2 Çoklu Depo Ortamının Oluşturulması

Çoklu depo sistemini oluşturmak için gerekli ilk gereksinim çok modelli dağıtık veri ortamının oluşturulmasıdır. Önceki bölümde MIMIC-III verisinin çözümlemesi yapılarak içerdiği verileri kapsayan modeller belirlenmişti. Dolayısıyla bir sonraki adım ise her veri parçasının kendisini kapsayan model yapısına ait depoya yüklenmesidir. Bu kapsamda çoklu depo ortamının kurulması adına yapılan işlemler bu bölümde detaylandırılacaktır. Bağlantılı olarak tez süreci boyunca ikinci danışmanım Prof. Dr. Oğuz DİKENELLİ yürütücülüğünde ve danışmanım Doç. Dr. Rıza Cenk ERDUR ve benim araştırmacı olarak yer aldığımız "Bir Çoklu Depo Ortamı Gerçekleştirimi" isimli bir bilimsel araştırma projesi (BAP) önerilmiş, başlatılmış, yürütülmüş ve tamamlanmıştır. Projedeki çalışmalar ışığında takip eden alt bölümlerde öncelikle veri depolarının seçilmesi ve bu seçimde göz önüne alınan kriterler

anlatılacak olup, sonrasında da MIMIC-III verisinin belirlenen depolara (saklama teknolojilerine) yüklenmesi adımları anlatılacaktır.

4.2.1 Veri Depolarının Seçilmesi

Bahsi geçen her veri modeli için kullanılacak veritabanı teknolojileri belirlenmiştir. Doğaldır ki belirlenen bu veriler ve çözümler zaman içerisinde evrilip değişebilecektir. Buna göre ilişkisel veri modeli için; PostgreSQL, zaman serisi veri modeli için; InfluxDB, anahtar-değer veri modeli için; Redis ve arama & belge veri modeli için de; Elasticsearch çözüm olarak seçilmiştir. Bu veritabanı çözümlerinin her birinin yönetimi ayrı ve derin bir uzmanlık gerektirmektedir. Tezde sunulan uygulamalı (applied) katkılardan biri de en yeni ve rekabetçi (state-of-the-art) NoSQL teknolojilerin entegre edilerek ortaya atılan savın çözümünde etkin rol oynadığını göstermektir. Dolayısıyla bahsi geçen bu teknolojileri daha etkin olarak küme bilgisayar ortamında çalıştırabilmek, bağımlılığını ve diğer sistemlere etkisini azaltmak adına Docker ve Kubernetes gibi çözümler de kurulum aşamasında denenip mümkün olduğunca kullanılmıştır.

4.2.1.1 İlişkisel Saklama Çözümü

İlişkisel veriler için kullanılan saklama çözümü olarak Postgresql[16] veritabanı belirlenmiştir. Buna göre Postgresql kendi deyimiyle dünyanın en gelişmiş açık kaynak ilişkisel veritabanı olarak ifade edilmektedir. C programlama dili ile yazılmış olup, 35 yıldan fazla süregelen ve devam eden bir aktif geliştirime dayanmasından ötürü çok büyük bir geliştirim topluluğuna sahiptir. Bu kadar çok tercih edilmesinin sebepleri ise sahip olduğu tutarlılık (reliability), sağlamlık (robustness) ve yüksek performans özellikleridir. Netflix, Uber, Twitch, Instagram, Spotify, Reddit ve bunun gibi birçok büyük ölçekli şirket tarafından kullanılıyor olması piyasadaki başarısını ve ilişkisel veritabanı ailesindeki önemini ortaya koymaktadır. Ayrıca kaynak kodu üzerinden

[16]https://www.postgresql.org

genişletilebilir olması bilimsel araştırma projelerinde kullanımını uygun hâle getirmektedir.

Tez kapsamında kurulan Postgresql veritabanındaki şemalardan birinde MIMIC-III verisinin çok fazla güncellenmeyen ve yüksek tutarlılık gerektiren alt kümesinin bulunmasının yanısıra, bir kıyaslama noktası oluşturmak amacıyla diğer bir şemada veri kümesinin tamamı yer almaktadır.

4.2.1.2 Zaman Serisi Saklama Çözümü

Zaman-serisi modelinde yer alacak veriler için saklama çözümü olarak zaman serisi veritabanları arasından InfluxDB[17] seçilmiştir. InfluxDB zaman serisi veriler için özelleşmiş ve GO[18] dili ile yazılmış açık kaynak bir veritabanıdır. Tez kapsamında kullanılan ve tek düğüm üzerinde çalışan versiyonu açık kaynak olmakla birlikte küme üzerinde çalışan ve bulut üzerinden sunulan çözümleri ücretlidir. Veri çözümleme ve izleme, sunucu metrikleri, IoT sensör verileri, borsa verileri, hastane yoğun bakım çizelge verileri vb. birçok önemli alandaki veriye uygulanabilmektedir. Rakibi olan diğer zaman serisi veritabanları arasında en iyi ve performanslı çalışan çözüm olarak karşımıza çıkmaktadır ve bu sebeple tezde kullanılmasına karar verilmiştir.

Veri yapısı ilişkisel veritabanına benzemekle birlikte 5 ana bileşenden oluşmaktadır:

1. Kova (Bucket): İlişkisel modeldeki veritabanına karşılık gelmektedir. Ölçümlerin düzenlenip saklandığı bir depo görevi görmektedir.
2. Ölçüm (Measurement): Veritabanındaki tablolara karşılık gelmektedir. Başlıca anahtar (primary key) zaman damgası (timestamp) olup ölçülen ilişkili değerlerin gruplanarak saklandığı bir yapı oluşturur.
3. Nokta (Point): Bir ilişkisel veritabanı tablosundaki satırlara karşılık gelir. Belirli bir zamandaki ölçüm değerini ifade etmektedir.
4. Alan (Field): Nokta'da yer alan ve ölçüm değeri ile birlikte tutulan

[17] https://www.influxdata.com/products/influxdb
[18] https://go.dev

verilerdir. Bir noktada birden çok alan yer alabilir. İlişkisel veritabanındaki indeks özelliği dışıdaki sütunlara karşılık gelmektedir. Bu veriler indekslenmez.

5. Etiketler (Tags): Noktada yer alan ve indekslenen verilerdir. İlişkisel veritabanındaki indeks özelliği olan sütunlara karşılık gelir.

Ölçüm verileri hat protokolü (line protocol) aracılığı ile HTTP, TCP veya UDP üzerinden yazılır. Flux[19], InfluxDB'den ve SQL Veritabanları gibi diğer birçok kaynaktan gelen verileri sorgulamak, işlemek, yazmak, analiz etmek ve bunlar üzerinde işlem yapmak için tasarlanmış güçlü ve fonksiyonel bir veri betik dilidir. Standart kütüphanesinde 400'e yakın fonskyion içermekte ve bu sayede veri üzerinde istenilen işlemleri yapmayı oldukça kolaylaştırmaktadır.

InfluxDB; Adobe, Cisco, Dell, IBM, ING Bank, Mozilla, Oracle, PayPal, RedHat, SAP, Siemens, SpaceX, Chicago Üniversitesi, ebay gibi önemli birçok kurum ve kuruluş tarafından geniş bir alan yelpazesinde kullanılmaktadır.

4.2.1.3 Anahtar-Değer Saklama Çözümü

Anahtar-değer yapısındaki verilerin saklanması için Redis[20] veritabanı seçilmiştir. Redis, belgelemesinde anlatıldığı üzere; veritabanı, önbellek, mesaj aracısı ve akış motoru olarak kullanılan bir açık kaynaklı (BSD lisanslı), bellek-içi veri yapısı deposudur. Redis, metinler (strings), karmalar (hashes), listeler, kümeler, sıralı kümeler gibi aralık (range) sorguları, bitmapler, hiperlogloglar, jeo-uzamsal indeksler ve akışlar gibi veri yapılarını destekler. Verileri hızlı bir şekilde okuma ve belleğe yazma yeteneği ile mükemmel performans sunabilmektedir.

Ek olarak, Redis atomik işlemleri desteklediğinden hızlı erişim süresine ihtiyaç duyulan önbelleğe alma senaryoları için idealdir. Bir metin verisinin sonuna ekleme, bir karmada saklanan değerin artırılması, bir listeye eleman ekleme, küme kesişimi, birleşimi ve farkı alabilme, sıralı bir dizideki en yüksek

[19] https://docs.influxdata.com/flux/v0.x

[20] https://redis.io

değerli üyeyi elde etme vb. gibi birçok atomik işlem yapılabilmektedir. Bunun dışında atomik işlemlerin yanısıra Lua[21] betimleme (Lua scripting) yöntemi ile doğrudan saklanan veri üzerinde anlık işlem gerçekleştirilebilmektedir.

En iyi performansı sunabilmek için Redis bellek-içi veri kümesi olarak hizmet verir. Tezde olduğu gibi kullanım durumuna bağlı olarak veriler diske yazılabilmektedir. Redis kalıcılığı (redis persistence), verilerin diske kaydedilmesini ve Redis sunucusu yeniden başlatıldığında geri yüklenmesini sağlayarak, bir çökme veya kapanma durumunda verilerin kaybolmamasını sağlar. Birden çok saklama tekniği olmakla birlikte, en çok ve varsayılan olarak kullanılanı Redis verisinin anlık görüntüsünün (snapshot) periyodik olarak diske yazılmasıdır.

Redis ANSI C ile yazılmış olup birçok LINUX ve benzeri sistem üzerinde dışsal ek bir gereksinim olmadan çalışabilmektedir. Twitter, GitHub, Snapchat, StackOverflow, Uber, Airbnb, Pinterest, Instagram, Udemy vb. birçok büyük kuruluş tarafından aktif olarak kullanılmaktadır.

4.2.1.4 Arama-Belge Saklama Çözümü

Metin odaklı veriler üzerinde anahtar kelimeler aracılığıyla aramaların yapılabilmesine olanak sağlamak ve eşleşen belgeleri getirebilmek için Elasticsearch [22] kullanılmaktadır. Elasticsearch; metinsel, sayısal, jeo-uzamsal, yapısal ve yapısal olmayan veriler de dahil olmak üzere her türlü veri üzerinde çalışabilen dağıtık ve açık kaynak bir arama ve analiz motorudur. Elasticsearch, Apache Lucene üzerine kurulmuş olup basit REST API'leri, dağıtık doğası (nature), hızı ve ölçeklenebilirliği ile tanınmaktadır.

Bilinen en gelişmiş arama altyapısını sunan Elasticsearch, ELK (Elasticsearch, Logstash, Kibana) yığıtı (stack) sayesinde arama özelliğinin dışında veri çözümleme, izleme ve görselleştirme gibi birçok farklı senaryo özelinde de kullanılabilmektedir.

Elasticsearch belge aramanın temelinde indeks mekanizması bulunmak-

[21] https://www.lua.org

[22] https://www.elastic.co/elasticsearch

tadır. Bir Elasticsearch indeksi, birbiriyle ilişkili belgelerin bir derlemi olarak ifade edilebilir. Elasticsearch, verileri JSON belgeleri olarak depolar. Her belge, bir dizi anahtarı (alanların veya özelliklerin adları) ile ilişkilendirilerek bunlara karşılık gelen değerlerle (metinler, sayılar, mantıksal değerler (booleans), tarihler, değer dizileri, coğrafi konumlar veya diğer veri türleri) birlikte saklanır.

Elasticsearch, ters indeks (inverted index) adı verilen ve çok hızlı tam metin aramalarına izin verecek şekilde tasarlanmış bir veri yapısı kullanır. Ters indeks, herhangi bir belgede yer alan tüm benzersiz (unique) kelimeleri listeler ve her bir kelimenin geçtiği tüm belgeleri tanımlar ve bulur. İndeks oluşturma işlemi sırasında, Elasticsearch belgeleri depolar ve belge verilerini neredeyse gerçek zamanlı olarak aranabilir hale getirmek için bir ters dizin oluşturur. Dizin oluşturma, belirli bir dizine bir JSON belgesi ekleyebileceğiniz veya güncelleyebileceğiniz dizin API'si ile başlatılır. Elasticsearch indeksi, sorgu apisi aracılığıyla JSON formatında sorgular yollanarak sorgulanabilmektdir.

Elasticsearch Java programlama dili ile geliştirilmiş olup, T-Mobile, Adobe, Booking.com, BMW, Jaguar, Land Rover, Cisco, Zurich sigorta vb. birçok büyük kuruluş tartafından kullanılmaktadır.

4.2.2 MIMIC-III Verisinin Seçilen Depolara Yüklenmesi

Bu bölümde veri kümesinin belirlenen parçalarının belirlenen depolara yüklenmesi aşamaları açıklanacaktır. Tüm veri yükleme işlemleri öncelikle yerel bilgisayar üzerinde gerçekleştirilmiştir. Depoların ve verilerin yerel bilgisayardaki varlığı test amaçlı olduğundan kalıcı yükleme yapılmamış olup, Docker ve/veya kaynak kod üzerinden yükleme tercih edilmiştir. Verilerin yerelde yüklendiğinden ve doğruluğundan emin olunduktan sonra depoların ve verilerin sunucu bilgisayarlar üzerine kurulumu kalıcı olacak şekilde tamamlanmıştır. Birinci sunucu (düğüm-1) üzerine Postgresql ve Elasticsearch, ikinci sunucuya (düğüm-2) da InfluxDB ve Redis doğrudan linux kütüphanesi yükleme adımları ile kurulmuştur.

MIMIC-III verisinin depolara yüklenmesi ve sorgulanması için Scala

dilinde DataImporter[23] isimli bir yükleme programı yazılmıştır. Buna göre her depo için yazılan yükleme programları içerisinde, ilgili depoyla bağdaştırılmış olan CSV formatındaki veriler Apache Spark kütüphanesi yardımıyla satır satır okunup işlenerek belirlenen depoya uygun biçime dönüştürülüp yüklenmektedir. Her deponun Scala istemci-bağdaştırıcı (client-connector) kütüphanesi araştırılmış, sonuç olarak mümkünse her depo tarafından resmi olarak önerilen, en çok kullanılan, belgelemesi, örnekleri ve etkinliği yüksek olan kütüphaneler belirlenmiştir.

Veri yükleme işlemine öncelikle ilişkisel veri deposu çözümü olan PostgreSQL ile başlanmıştır. MIMIC-III[24] kod tabanının yer aldığı sayfada PostgreSQL ve birçok farklı popüler veritabanı çözümü üzerine CSV formatındaki MIMIC verisinin yüklenmesi aşamaları detaylıca anlatılmaktadır. Buna göre hem veritabanının yerele kurulumu hem de verinin yükleme adımları doğrudan docker üzerinden takip edilerek tüm veri "mimic" isimli veritabanına yüklenmiştir. Daha sonra ilişkisel depoya ayrıştırılan veriler kopyalanarak yine postgres içinde "poly_mimic_rel" adı altında ayrı bir veritabanına taşınmıştır. Böylece ilişkisel verinin yere depo üzerine yüklenmesi işlemi tamamlanmıştır. Listeleme-1 ve 2'de MIMIC-III verisinin Docker üzerinden Postgresql bireyine (instance) yükleme adımları görülmektedir. Listeleme-1 MIMIC-III verisinin Postgresql üzerinde çalışan görüntüsünün (image) oluşturulabilmesi için gerekli adımı göstermektedir. Bunun için öncelikle "csv.gz" uzantılı MIMIC-III dosyalarının yüklenebilmesi için "/mimic_code/buildmimic/docker/mimic_data" isimli klasörün altına kopyalanması gerekmektedir. Sonrasında Liseleme-1'deki kabuk betiği (shell script) işletilerek Postgresql'in çalışmaya ve MIMIC-III verisini yüklemeye hazır görüntüsü elde edilir.

```
1 cd /mimic_code/buildmimic/docker
2 docker build -t postgres/mimic .
```

Listeleme 1: MIMIC-III yükleme için Postgres Docker Görüntüsünün Oluşturulması

[23] https://github.com/burakyonyul/data-importer.git
[24] https://github.com/MIT-LCP/mimic-code/tree/main/mimic-iii

Daha sonra oluşturulan bu Postgresql görüntüsü Listeleme-2'deki dağıtım (deploy) kabuk betiği işletilerek Docker üzerinde çalıştırılmış olur ve MIMIC-III veritabanı da belirtilen klasördeki csv dosyaları otomatik olarak yüklenerek kurulur. Tüm MIMIC-III verisi tablolar halinde "mimic" isimli veritabanı içerisindeki mimiciii şemasına kopyalanır ve gerekli indeksler oluşturulur.

```
docker run \
--name mimic \
-p HOST_PORT:5432 \
-e BUILD_MIMIC=1 \
-e POSTGRES_PASSWORD=POSTGRES_USER_PASSWORD \
-e MIMIC_PASSWORD=MIMIC_USER_PASSWORD \
-v /HOST/mimic_data/csv:/mimic_data \
-v /HOST/PGDATA_DIR:/var/lib/postgresql/data \
-d postgres/mimic
```

Listeleme 2: MIMIC-III için Postgres'e Yüklenmiş Docker Görüntüsünün Çalıştırılması

Bunun dışında sunucu üzerine veri yüklemek için yine MIMIC-III kod tabanındaki adımlar takip edilerek sql boşaltma (dump) yöntemiyle düğüm-1 üzerinde kurulmuş ve çalışmakta olan Postgresql veritabanına yerelde yapıldığı şekilde tüm veri ve ayrıştırılan ilişkisel veriler sırasıyla "mimic" ve "poly_mimic_rel" veritabanlarına yüklenmiştir. İndirilen "mimic_code" klasörüne kabuk üzerinden giriş yaptıktan sonra Listeleme-3'te görülen bilindik (standard) yükleme adımları işletilerek verinin yüklenmesi tamamlanır. Buradaki "datadir" MIMIC-III verisinin bulunduğu "csv.gz" dosyalarının bulunduğu klasörü ifade eder.

```
make create-user mimic datadir="/path/to/data/"
```

Listeleme 3: MIMIC-III verisinin Postgres'e Yüklenmesi

"make" komutu varsayılan olarak aşağıdaki parametreleri kullanmaktadır:

- Veritabanı Adı: mimic
- Kullanıcı Adı: postgres

- Şifre: postgres
- Şema: mimiciii
- Sunucu: localhost
- Port: 5432

MIMIC-III veri kümesinin belgelemesinde bahsedildiği üzere genel sorgulama ve veri erişimi bakış açısından bütünsel olarak verilerinin temsil edildiği ve sunulduğu veri modeli ilişkisel model, önerilen veritabanı da Postgresql'dir. Dolayısıyla bu verilerin Postgres'e yüklenmesi için hazır Docker görüntüsü, sql boşaltma (dump) vb. gibi birçok yöntem olduğundan ilişkisel verilerin yüklemesi kod aracılığı ile yapılmamış olup, yalnızca bağlantı ve sorgulama ile ilgili kod yazılmıştır. Listeleme-4'te (listing) yüklenen Postgres verilerinin sorgulanması ile ilgili örnek kod parçacığı görülmektedir. Burada değerlendirme bölümünde anlatılan taslak sorgulardan olan; sepsis teşhisi konan ve notlarında "azotemia" geçen hastaların listelenmesi ile ilgili sorgulama işlemi görülmektedir.

```
import java.sql.ResultSet
object PostgresqlQuerier extends App {
  val start = System.nanoTime()
  val con_st = "jdbc:postgresql://localhost:5432/mimic?user=postgres&password=admin123"
  val conn = PostgresqlDataSource.hikariDataSource.getConnection
  try {
    val stm = conn.createStatement(
      ResultSet.TYPE_FORWARD_ONLY,
      ResultSet.CONCUR_READ_ONLY)
    val sqlQuery =
      s"""
          SELECT * FROM mimiciii.noteevents
          INNER JOIN mimiciii.diagnoses_icd
          ON noteevents.subject_id=diagnoses_icd.subject_id
          WHERE LOWER(noteevents.text) LIKE '%azotemia%'
          AND diagnoses_icd.icd9_code
          IN ('99591','99592','77181','67020','67022','67024')
          """.stripMargin
    val rs = stm.executeQuery(sqlQuery)
    val metaData = rs.getMetaData
    while (rs.next) {
      for (columnNo <- 1 to metaData.getColumnCount) {
        val columnLabel = metaData.getColumnLabel(columnNo)
        val columnValue = rs.getString(columnNo)
        print(s"$columnLabel:$columnValue, ")
      }
      println()
    }
    val duration = (System.nanoTime() - start) / 1e9d
    println(s"Elapsed seconds: $duration")
  } finally {
```

```
32      conn.close()
33    }
34  }
```

Listeleme 4: Postgres'e Yüklenen Veriyi Sorgulayan Örnek Scala Kodu

Veri yükleme işlemine anahtar-değer veri deposu olan Redis ile devam edilmiştir. Redis veritabanı da benzer olarak yerelde docker üzerinden kurulmuştur. Scala dilindeki istemci bağdaştırıcı (Scala-Redis client connector)[25] aracılığı ile bağlantı kurulup CSV verilerinin [anahtar - değerler listesi] formuna dönüştürülerek Redis'e yüklenmesi sağlanmıştır. Listeleme-5'te Redis'e yükleme ve yüklenen verilerin sorgulanması için oluşturulan koddan öğeler sözlüğünün (D_items) yüklenmesine ilişkin kod parçası görülmektedir. "main" metodu üzerinden import_d_items() çağrılarak öğeler sözlüğünün yükleme tetiklemesi yapılır (*satır:* **36**). Metodunun açıklamasında ilgili tablodaki verilerin Redis'te tutulduğu veri yapısı da görülmektedir. Bunda göre "itemid" sahası anahtar olarak seçilmiş ve tablodaki geri kalan sahalar değer kısmında liste tipinde tutulmaktadır. Ters indeks (inverted index) olarak da etiket değerine karşılık gelen öğeler listesi "label" anahtarı üzerinden erişilmektedir. Metodun içinde öncelikle öğeler sözlüğü için redis veritabanı belirlenerek "0" indeksli veritabanı seçilir (*satır:* **26**). Anahtarlara karşılık saklanan değerlerin karışmaması için her tablo ayrı indekste yer alan bir veritabanına kaydedilmiştir. Daha sonra veritabanının içeriği temizlenerek eskiden kaydedilen anahtarların karmaşa yaratma olasılığından kaçınılmış olur (*satır:* **27**). Apache SPARK kütüphanesinin CSV okuma metodu ile D_ITEMS için gerekli dosya yolu sağlanarak okuma işlemi başlatılır (*satır:* **28**). SPARK'ın filtreleme metotlarıyla null değerlerin ayıklanması, sütun açıklama satırından kaçınılması vb. veri temizliği yapıldıktan sonra okunan her csv satırının Redis'te seçilen veritabanına yüklenmesini sağlayan insert_d_items() metodu çağrılır (*satır:* **30**). Bu metodun içinde öncelikle ilgili CSV satırındaki "itemid" ve "label" anahtar değerleri elde edilir (*satır:* **12, 13**). Daha sonra bu anahtarlar üzerinde saklanacak değerler listeye sağdan itme (rpush) metodu ile eklenir

[25] https://github.com/debasishg/scala-redis

(*satır:* **14, 18**). Son olarak tüm veriler yüklendikten sonra import_d_items() metodunda örnek anahtarlar sorgulanarak doğrulaması yapılır (*satır:* **32**). Bilinen bir anahtardaki değerler eklenen sırada elde edilirken itmenin yapıldığı sıranın tersinde sorgulanmalıdır. Yani sağdan itme yöntemiyle ekleme (rpush) yapıldığı için soldan mesafe (lrange) yöntemiyle sorgulama yapılır. Bilinmeyen ancak arama yapılan anahtar değerleri de keys metoduna yollanan düzenli ifade (regular expression) ile sorgulanabilmektedir (*satır:* **33**).

```
1  import com.redis.RedisClient
2  import org.apache.spark.sql.{Row, SparkSession}
3  object RedisImporter {
4    val redisFolderPrefix =
5      "/Users/burakyonyul/Development/Datasets/mimic-partitioned/key-value/"
6    val redisClient = new RedisClient("0.0.0.0", 6379)
7    var index = 0
8    val spark: SparkSession = SparkSession
9      .builder().appName("Spark CSV Reader")
10     .config("spark.master", "local").getOrCreate()
11   private def insert_d_items(row: Row): Unit = {
12     val labelValue = row.get(2)
13     val itemId = row.get(1)
14     redisClient.rpush(itemId,
15       labelValue, row.get(3), row.get(4), row.get(5),
16       row.get(6), row.get(7), row.get(8), row.get(9))
17     if (labelValue != null && labelValue.toString.nonEmpty)
18       redisClient.rpush(labelValue, itemId)
19   }
20   /**
21    * ITEMID --> [LABEL, ABBREVIATION, DBSOURCE, LINKSTO
22    * , CATEGORY, UNITNAME, PARAM_TYPE, CONCEPTID]
23    * LABEL --> [ITEMID]
24    */
25   private def import_d_items(): Unit = {
26     redisClient.select(0)
27     redisClient.flushdb
28     val df = spark.read.csv(redisFolderPrefix + "D_ITEMS.csv")
29     val head = df.head()
30     df.na.fill("").filter(row => row != head).foreach(row => insert_d_items(row))
31     insert_missing_d_items()
32     println(redisClient.lrange("500", 0, 7))
33     println(redisClient.keys("*Arterial*"))
34   }
35   def main(args: Array[String]): Unit = {
36     import_d_items()
37     import_d_lab_items()
38     import_d_icd_procedures()
39     import_d_icd_diagnoses()
40     import_caregivers()
41     import_procedures_icd()
42     import_diagnoses_icd()
43   }
44 }
```

Listeleme 5: Redis'e Yüklenen Veriyi Sorgulayan Örnek Scala Kodu

Üçüncü olarak zaman serisi verilerin InfluxDB'ye yüklenmesi gerçekleşti-

rilmiştir. Başta PostgreSQL ve Redis'e benzer olarak InfluxDB de Docker servis olarak yerel bilgisayar üzerine kurulmuştur. Veri yükleme işlemi de yine Scala dilinde, InfluxDB client connector aracılığıyla ve Apache Spark ile işlenerek gerçekleştirilmiştir. Fakat veri yükleme işlemi denenirken Docker üzerine kurulan InfluxDB servisi her yüksek boyuta ulaştığında kendini kapatmaktaydı. Sorunun çözümü için çok sayıda yükleme denemesi yapılıp uzun uğraşlar verildikten sonra Docker servisinden vazgeçilmiş, InfluxDB doğrudan yerel bilgisayar üzerine kurulmuştur. Sonuç olarak servisin kapanma sorunu ortadan kalkmış ve yaklaşık 400 milyon kayıttan (tuples) oluşan zaman serisi verilerin yüklenmesi tamamlanmıştır. Veri yoğunluğu çok fazla olduğundan tamamlanması yaklaşık 1 hafta kadar sürmüştür. Daha sonrasında ise doğrudan InfluxDB'nin yerel bilgisayar üzerindeki veri (data) klasörü düğüm-2 altında kurulmuş olan InfluxDB'nin veri klasörüne kopyalanarak verilerin sunucu üzerinde kurulumu ve erişimi tamamlanmıştır. Listeleme-6'da InfluxDB'ye hasta çizelge verilerini içeren CHARTEVENTS'in yüklenmesi adımları görülmektedir. InfluxDB'ye yükleme adımları Redis'teki yükleme adımlarına benzer olarak işletilmektedir. "main" metodunda "bulkImportChartEvents()" metodu çağrılarak hasta yatış verilerinin yüklemesi işlemi başlatılır (*satır:* **121**). Metodun adından da anlaşılacağı üzere yoğun verinin makul süreler içerisinde etkin şekilde yönetilebilmesi için toplu (bulk) yükleme yapılmaktadır. SPARK aracılığıyla CHARTEVENTS tablosundaki veriler okunarak veri filtreleme yapılır (*satır:* **44**). Sonrasında "bulkInsertChartEvents()" metodu ile okunan her satır tampona (buffer) eklenerek (*satır:* **64**) belirlenen limite ulaştıktan sonra "writePointsInBuffer()" metodu ile InfluxDB sunucusundaki "mimic-iii" isimli kovaya (bucket) yazılır (*satır:* **66**). CSV'deki her çizelge satırı okunduğunda bir nokta (point) yaratılarak (*satır:* **62**) kayıtlı değer kova içerisinde "chartevent" ölçümü altına zaman damgası ile birlikte eklenir (*satır:* **55**). Tüm veri için aynı adımlar işletilerek yükleme tamamlanır. Chartevents verisi içerdiği yaklaşık 330 milyon kayıt ile MIMIC-III'teki verilerin büyük çoğunluğunu oluşturmaktadır. Hasta gizliliğini korumak amacıyla MIMIC-III'te yapılan kimliksizleştirme sonucu zaman damgalı verilerin hepsi yıl

bazında değiştirilmiş olup bazı sapmalar dışında ”2100 - 2209” aralığına ötelenmiştir. Dolayısıyla veri üzerinde mantıklı ve makul zaman aralığı sorguları atılabilmesi için bu tarihlerin belirli bir mantıkta mümkün olduğunca orjinaline yakın biçime getirilmesi gerekmiştir. Bu sebeple InfluxDB'ye yüklenen tüm kayıtların zaman içeren verileri yıl bazında dönüştürülerek gerçek yıl aralığı olan ”2001-2012” arasına normalize edilmiştir. Dönüştürülen tarihlerle birlikte orjinal tarihler de saklanmış, böylece farklı veri modeleri arasında tarih üzerinden birleşim yapılması ve tutarlılığın korunması mümkün kılınmıştır.

```
import akka.actor.ActorSystem
import akka.stream.scaladsl.{Keep, Sink, Source}
import com.github.nscala_time.time.Imports.{DateTime, DateTimeFormat}
import com.influxdb.client.domain.WritePrecision
import com.influxdb.client.scala.InfluxDBClientScalaFactory
import com.influxdb.client.write.Point
import com.influxdb.query.FluxRecord
import org.apache.spark.sql.{Row, SparkSession}
import scala.collection.mutable.ListBuffer
import scala.concurrent.Await
import scala.concurrent.duration.Duration

object InfluxdbImporter {
  // in general except from outliers min year --> 2100, max year --> 2209
  private val ALTERED_MIN_YEAR = 2100
  private val ALTERED_MAX_YEAR = 2209
  private val MIN_YEAR_MIMIC = 2001
  private val MAX_YEAR_MIMIC = 2012
  private val BULK_BUFFER_SIZE = 5000
  private val WAIT_RECORD_LIMIT = 5000
  private val BATCH_RECORD_LIMIT = 500000000
  private var currentRowCount = 0
  private val DATE_PATTERN = "yyyy-MM-dd HH:mm:ss"
  val influxdbFolderPrefix =
  "/Users/burakyonyul/Development/Datasets/mimic-partitioned/time-series/"
  val token =
  "_g0QMff1wsKFHft1sf6rFs4VwKgFYINv7DGHiGlRGzlI5-Kd70rUEk4swdohDJYlf29OMbvv4grDyV-uR8sqSw==
    ="
  val org = "EgeUniversity"
  val bucket = "mimic-iii"
  val serverURL = "http://localhost:8086"
  val client = InfluxDBClientScalaFactory.create(serverURL, token.toCharArray, org, bucket
    )
  implicit val system: ActorSystem = ActorSystem("mimic-iii-data-loader")
  val spark: SparkSession = SparkSession
    .builder()
    .appName("Spark CSV Reader")
    .config("spark.master", "local")
    .getOrCreate()

  private def bulkImportChartEvents(): Unit = {
    val df = spark.read.csv(influxdbFolderPrefix + "CHARTEVENTS.csv")
    val head = df.head()
    val pointBuffer = new ListBuffer[String]()
    val start = System.nanoTime()
    df.na.fill("").filter(row => row != head).foreach(
    row => bulkInsertChartEvents(row, pointBuffer))
    val elapsed = (System.nanoTime() - start) / scala.math.pow(10, 9)
```

```
47      println(s"Elapsed time: $elapsed seconds")
48      if (pointBuffer.nonEmpty) {
49        writePointsInBuffer(pointBuffer)
50      }
51    }
52
53    private def writePointsInBuffer(lineBuffer: ListBuffer[String]): Unit = {
54      val source = Source.single(lineBuffer.toList)
55      val sink = client.getWriteScalaApi.writeRecords()
56      val materialized = source.toMat(sink)(Keep.right)
57      Await.ready(materialized.run(), Duration.Inf)
58      lineBuffer.clear()
59    }
60
61    private def bulkInsertChartEvents(row: Row, lineBuffer: ListBuffer[String]): Unit = {
62      val point: Point = createPointForChartEvent(row)
63      if (lineBuffer.size < BULK_BUFFER_SIZE) {
64        lineBuffer += point.toLineProtocol
65      } else {
66        writePointsInBuffer(lineBuffer)
67        currentRowCount += BULK_BUFFER_SIZE
68        println(s"Current row count: $currentRowCount")
69        if (currentRowCount % WAIT_RECORD_LIMIT == 0) {
70          Thread.sleep(1000)
71        }
72        if (currentRowCount % BATCH_RECORD_LIMIT == 0) {
73          Thread.sleep(30 * 60 * 1000)
74        }
75      }
76    }
77
78    private def createPointForChartEvent(row: Row): Point = {
79      val mimicNormalizer = MinMaxNormalizer(MIN_YEAR_MIMIC, MAX_YEAR_MIMIC)
80      val existingChartTime = DateTime.parse(
81      row.get(5).toString, DateTimeFormat.forPattern(DATE_PATTERN))
82      val normalizedChartTime = existingChartTime.withYear(mimicNormalizer.normalize(
83      ALTERED_MIN_YEAR, ALTERED_MAX_YEAR, existingChartTime.getYear).toInt)
84      val existingStoreTime = if (row.get(6).toString.nonEmpty)
85      DateTime.parse(row.get(6).toString, DateTimeFormat.forPattern(DATE_PATTERN))
86      else ""
87      val normalizedStoreTime =
88        if (row.get(6).toString.nonEmpty) {
89          existingStoreTime.asInstanceOf[DateTime].withYear(
90          mimicNormalizer.normalize(ALTERED_MIN_YEAR, ALTERED_MAX_YEAR,
91          existingStoreTime.asInstanceOf[DateTime].getYear).toInt)
92        } else {
93          ""
94        }
95
96      val point = Point
97        .measurement("chart_event")
98        .addTag("subject_id", row.get(1).toString)
99        .addTag("itemid", row.get(4).toString)
100       .addField("hadm_id", row.get(2).toString)
101       .addField("icustay_id", row.get(3).toString)
102       .addField("charttime", normalizedChartTime.toString)
103       .addField("deidentifiedcharttime", existingChartTime.toString)
104       .addField("storetime", normalizedStoreTime.toString)
105       .addField("deidentifiedstoretime", existingStoreTime.toString)
106       .addField("cgid", row.get(7).toString)
107       .addField("value", row.get(8).toString)
108       .addField("valuenum", row.get(9).toString)
109       .addField("valueom", row.get(10).toString)
110       .addField("warning", row.get(11).toString)
```

```
111       .addField("error", row.get(12).toString)
112       .addField("resultstatus", row.get(13).toString)
113       .addField("stopped", row.get(14).toString)
114       .time(normalizedChartTime.toDate.toInstant, WritePrecision.NS)
115     point
116   }
117
118   def main(args: Array[String]): Unit = {
119     if (args.nonEmpty) {
120       args(0) match {
121         case "chartevents" => bulkImportChartEvents()
122         case "datetimeevents" => bulkImportDatetimeEvents()
123         case "procedureevents" => bulkImportProcedureEvents()
124         case "inputeventscv" => bulkImportInputEventsCV()
125         case "inputeventsmv" => bulkImportInputEventsMV()
126         case "labevents" => bulkImportLabEvents()
127         case "outputevents" => bulkImportOutputEvents()
128         case "microbiologyevents" => bulkImportMicrobiologyEvents()
129       }
130     } else {
131       println("No import data is given, please provide a label to import this data.")
132     }
133     queryExampleData
134     client.close()
135     system.terminate()
136   }
137 }
```

Listeleme 6: InfluxDB'ye Veri Yükleyen ve Sorgulayan Örnek Scala Kodu

Son olarak ise arama-belge tipi verilerin Elasticsearch üzerine yüklenmesi gerçekleştirilmiştir. Benzer olarak Elasticsearch başlangıçta Docker servis olarak açılmış, fakat InfluxDB'de yaşanan problemler göz önünde bulundurularak doğrudan sıkıştırılmış biçimde (tar.gz) yerel bilgisayara indirilerek kurulmuştur. Noteevents tablosundaki "text" sahasında yer alan ham metin bilgileri CSV dosyasından okunurken bazı özel karakterlerden ötürü sorunlar çıktığından doğrudan PostgreSQL üzerindeki "mimic" veritabanından parça parça sorgulanıp dönüştürülerek elastic4s [26] kütüphanesi aracılığıyla Elasticsearch üzerine yüklenmiştir. Böylece çoklu depo ortamını oluşturan tüm depolar servis olarak kurulmuş, içerisine belirlenen veriler yüklenmiş ve sorgulanmaya hazırdır. Listeleme-7'de Noteevents tablosunun csv biçimindeki verisinin Spark kütüphanesi ile okunup dönüştürülerek Elasticsearch'e yüklenmesi işlemlerini gerçekleştiren Scala kodu görülmektedir. Elasticsearch servisi üzerinde saklanacak olan veriler belirlenen indeksler altında tutulmaktadır. Dolayısıyla

[26] https://github.com/sksamuel/elastic4s.git

MIMIC-III içerisindeki metin yoğun veriler NOTEEVENTS tablosunda bulunduğundan, yalnızca bu tablonun verileri Elasticsearch üzerinde yine aynı isimdeki "noteevents" indeksi altında saklanmaktadır. İlk olarak "removeNoteEventsIndex()" metodu ile indeks altında kayıtlı veriler silinmektedir (*satır:* **133**). Daha sonra "createNoteEventsIndex()" metodu ile Elasticsearch'te saklanacak verinin yapısı tanımlanarak "noteevents" isimli indeks yaratılır (*satır:* **134**). NOTEEVENTS tablosundaki "text" sahası metin verisi içerdiğinden yaratılan Elasticsearch indeksinde text alanı (textField) olarak tanımlanmış olup tam metin aramasına (full text search) olanak tanımaktadır (*satır:* **84**). Diğer sahalar anahtar alan (keywordField) olarak tanımlanmış olup yalnızca bire bir eşleme (exact match) yapılabilmektedir. İndeks yaratıldıktan sonra "importNoteEventsIndex()" metodu çağrılarak veri yükleme işlemi tetiklenir (*satır:* **135**). Burada postgresql üzerinde tüm veriyi içeren "mimic" şemasına bağlanılarak NOTEEVENTS tablosundaki veri 5000'er satırlık parçalar halinde sorgulanıp her satır için kurulan yapıda ayrı ayrı "noteevents" indeks isteği nesnesi oluşturulup tampona eklenir (*satır:* **51**). Tampon 5000 istekle dolduktan sonra bunlar "writeNoteEventsFromBuffer()" metodu ile topluca işletilerek Elasticsearch'e yazılır (*satır:* **54**). Tablodaki 2083180 kayıt bu şekilde işlenerek verinin yüklenmesi tamamlanır. Son olarak "queryElasticSearch()" metodu ile "noteevents" indeksine yazılan veriler "text" alanında "azotemia" anahtar kelimesini içeren belgeleri (documents) elde edecek şekilde test amaçlı sorgulanmaktadır (*satır:* **136**).

```
1
2  import com.sksamuel.elastic4s.http.JavaClient
3  import com.sksamuel.elastic4s.requests.common.RefreshPolicy
4  import com.sksamuel.elastic4s.requests.indexes.IndexRequest
5  import com.sksamuel.elastic4s.requests.searches.SearchResponse
6  import com.sksamuel.elastic4s.{ElasticClient, ElasticProperties,
7  RequestFailure, RequestSuccess}
8
9  import java.sql.{DriverManager, ResultSet}
10 import scala.collection.mutable.ListBuffer
11
12 object ElasticsearchImporterPostgres extends App {
13   private val MAX_ROW_COUNT = 2083180
14   private val BULK_LIMIT = 5000
15   val props = ElasticProperties("http://127.0.0.1:9200")
16   val client = ElasticClient(JavaClient(props))
17   private var currentRowIndex = 0
18   classOf[org.postgresql.Driver]
19   val con_st = "jdbc:postgresql://localhost:5432/mimic?user=postgres&password=admin123"
```

```
20  import com.sksamuel.elastic4s.ElasticDsl._
21
22  def createNoteEventsIndex(): Unit = {
23    client.execute {
24      createIndex("noteevents").mapping(
25        properties(
26          keywordField("subject_id"),
27          keywordField("hadm_id"),
28          keywordField("chartdate"),
29          keywordField("charttime"),
30          keywordField("storetime"),
31          keywordField("category"),
32          keywordField("description"),
33          keywordField("cgid"),
34          keywordField("iserror"),
35          textField("text")
36        )
37      )
38    }.await
39  }
40
41  private def importNoteEvents(): Unit = {
42    val conn = DriverManager.getConnection(con_st)
43    val indexReqBuffer = new ListBuffer[IndexRequest]()
44    val start = System.nanoTime()
45    try {
46      while (currentRowIndex < MAX_ROW_COUNT) {
47        val stm = conn.createStatement(ResultSet.TYPE_FORWARD_ONLY, ResultSet.
    CONCUR_READ_ONLY)
48        val rs = stm.executeQuery(
49        s"SELECT * from mimiciii.noteevents LIMIT $BULK_LIMIT OFFSET $currentRowIndex")
50        while (rs.next) {
51          indexReqBuffer += createNoteEventIndexRequest(rs)
52        }
53        currentRowIndex += BULK_LIMIT
54        writeNoteEventsFromBuffer(indexReqBuffer)
55        println(s"Current row index: $currentRowIndex")
56      }
57      val elapsed = (System.nanoTime() - start) / scala.math.pow(10, 9)
58      println(s"Elapsed time: $elapsed seconds")
59    } finally {
60      conn.close()
61    }
62  }
63
64  private def writeNoteEventsFromBuffer(indexReqBuffer: ListBuffer[IndexRequest]): Unit =
      {
65    client.execute {
66      bulk(
67        indexReqBuffer
68      ).refresh(RefreshPolicy.Immediate)
69    }.await
70    indexReqBuffer.clear()
71  }
72
73  private def createNoteEventIndexRequest(row: ResultSet): IndexRequest = {
74    val indexReq = indexInto("noteevents").id(row.getString("row_id")).fields(
75      "subject_id" -> row.getString("subject_id"),
76      "hadm_id" -> row.getString("hadm_id"),
77      "chartdate" -> row.getString("chartdate"),
78      "charttime" -> row.getString("charttime"),
79      "storetime" -> row.getString("storetime"),
80      "category" -> row.getString("category"),
81      "description" -> row.getString("description"),
```

```
82        "cgid" -> row.getString("cgid"),
83        "iserror" -> row.getString("iserror"),
84        "text" -> row.getString("text")
85      )
86      indexReq
87    }
88
89    def queryElasticSearch(): Unit = {
90      val start = System.nanoTime()
91      val resp = client.execute {
92        search("noteevents").rawQuery(
93          s"""
94             |{
95             |    "bool" : {
96             |      "must" : [
97             |        {
98             |            "query_string": {
99             |              "query": "azotemia",
100            |              "default_field": "text"
101            |            }
102            |        }
103            |      ]
104            |    }
105            |},
106            |"size":10000
107            |""".stripMargin
108       )
109     }.await
110     println("---- Search Results ----")
111     var count = 0
112     resp match {
113       case failure: RequestFailure => println("We failed " + failure.error)
114       case results: RequestSuccess[SearchResponse] =>
115         val searchHits = results.result.hits.hits.toList
116         searchHits.foreach { searchHit =>
117           println(s"${searchHit.id} -> ${searchHit.sourceAsMap}")
118           count = count + 1
119         }
120       case results: RequestSuccess[_] => println(results.result)
121     }
122     println(count)
123     resp foreach (search => println(s"There were ${search.totalHits} total hits"))
124     val duration = (System.nanoTime() - start) / 1e9d
125     println(s"Elapsed seconds: $duration")
126   }
127   def removeNoteEventsIndex(): Unit = {
128     client.execute {
129       deleteIndex("noteevents")
130     }.await
131   }
132
133   removeNoteEventsIndex()
134   createNoteEventsIndex()
135   importNoteEvents()
136   queryElasticSearch()
137   client.close()
138 }
```

Listeleme 7: Elasticsearch'e Veri Yükleyen ve Sorgulayan Örnek Scala Kodu

4.3 Ölçeklenebilir Sorgu İşleme ve Birleşim Mekanizması

NoSQL veritabanları ve büyük veri işleme araçları bilindik yöntemlerle yönetilemeyecek yapıda, hacimde ve yoğunluktaki verinin saklanmasını, işlenmesini ve sorgulanmasını kolaylaştırmak için ortaya çıkmışlardır. Bunun yanında bahsi geçen veri birden fazla kaynaktan geliyorsa ve bunlar birbiriyle bağlantılı ise yapısı gittikçe karmaşıklaşmakta ve bütünleştirilmesi gerekmektedir. Çoklu depo sistemleri bünyesinde saklama çözümü olarak ilişkisel ve NoSQL veritabanlarını bulundurmasının yanında, içerdiği sorgu motoru ve veri bütünleştirme yöntemi ile de bu çok modelli veri işlenmesine olanak sunmaktadır. Büyük veriyi çoklu depo sistemleri ile yönetebilmek (big data management) için çok modelli dağıtık veri ortamının kurulumundan sonra bu verinin bütünleştirilip sorgulanabilmesi için sistemin temelinde yer alacak ölçeklenebilir bir sorgu işleme mimarisinin tasarlanması gerekmektedir. Aynı zamanda sorgu sonucunu oluşturmak için kullanılacak birleşim algoritmasının da ölçeklenebilirliği destekleyecek ve büyük miktardaki veriyi işlemeye uygun yapıda olması gerekmektedir.

Alan odaklı dağıtık çoklu depo ortamının oluşturulmasından sonra, tezin bir diğer ana katkı noktası da çoklu depo sisteminin "sorgu işleme" (query processing) altyapısıdır. Sorgu işlemenin temel mantığı ara sonuçları kullanarak ana sonucun elde edilmesi yöntemidir. Bunun için ilişkisel veritabanı literatüründeki en eski ve geçerli yöntemlerden biri birleşim (join) tekniğidir. Sistemin en önemli gereksinimleri sorguların performanslı, eşzamanlı (concurrent), paralel ve ölçeklenebilir işletimi olarak belirlenmiştir. Bu durumda ana sorguların yanı sıra, ana sorguları oluşturan alt-sorguların birbirinden bağımsız işletilmesi eşzamanlılık ve paralellik açısından elzemdir.

Dolayısıyla öncelikle sorgu işleme yöntemi belirlenmeli ve mümkünse bu yöntemin etkin çözümlerinden birisi seçilerek kullanılmalı, daha sonra seçilen sorgu işleme altyapısıyla uyumlu çalışabilecek etkin bir birleşim algoritması seçilerek gerçekleştirilmelidir. MIMIC-III veri kümesinin çözümlemesi sonucu

elde edilecek sınama sorgu kümesiyle de sorgu işleme ve birleşim mekanizmasının başarımı sınanabilecektir.

4.3.1 Sorgu İşleme Altyapısının Belirlenmesi

Büyük veri işleme araçları her geçen gün daha popüler hâle gelmekte, böylece rekabetçi ve alternatifli bir seçim kümesi oluşmaktadır. Dolayısıyla geliştirilen sistemde paralel, eş zamanlı ve dağıtık bir sorgu işleme altyapısı seçilerek kurulacak çoklu depo sisteminin etkinliğini destekleyecek bir mimari ortaya konmalıdır. Böyle bir teknik sistemin çekirdeğini oluşturacağından çoklu depo sisteminin büyük veri yönetimi gereksimlerini karşılamalı ve sistemin bütünüyle uyum içinde çalışmalıdır.

Bu kapsamda piyasada veri akışı işleme bakış açısından Apache Yazılım Kuruluşu (Apache Software Foundation) altında Spark [27], Kafka [28], Flink [29], Samza [30], Pulsar [31] gibi açık kaynak çözümler, ücretli olarak sunulan Amazon Kinesis [32] ve Google Cloud Dataflow [33] servisleri, ücretli ve ücretsiz sürümleri olan HazelCast [34], yeni ve popülerlik kazanan Spring Cloud Data Flow [35] vb. birçok araç bulunmaktadır. Diğer taraftan paralel ve dağıtık hesaplama bakış açısından açık kaynak olarak Apache Yazılım Kuruluşu altındaki Hadoop [36] ve Storm [37] çerçevelerinin (framework) yanında, aktör model çerçevesi (actor model framework) üzerinden çözüm sunan Akka [38], Riker [39], CAF

[27] https://spark.apache.org
[28] https://kafka.apache.org
[29] https://flink.apache.org
[30] https://samza.apache.org
[31] https://pulsar.apache.org
[32] https://aws.amazon.com/kinesis
[33] https://cloud.google.com/dataflow
[34] https://hazelcast.com
[35] https://spring.io/projects/spring-cloud-dataflow
[36] https://hadoop.apache.org
[37] https://storm.apache.org
[38] https://akka.io
[39] https://riker.rs

[40], Comedy [41], Orleans [42] vb. birçok kütüphane bulunmaktadır. Erlang [43] ve Elixir [44] programlama dilleri de hafif (lightweight) iş parçacığına sahip Erlang çalışma zamanı (runtime) üzerine inşa edildiklerinden ve yerleşik (built in) aktör programlama özelliğine sahip olduklarından yüksek seviyede dağıtık ve ölçeklenebilir uygulamaların geliştirilmesine olanak sağlamaktadır.

Tüm bu çerçevelerin özellikleri, güçlü ve zayıf yanları tartıya konularak, geliştirilecek çoklu depo sisteminin de gereksinimleri göz önünde bulundurulduğunda sorgu işleme altyapısının, ölçeklenebilir, dağıtık (distributed), paralel ve eş zamanlı (concurrent) çalışabilme özelliklerinin hepsini bünyesinde barındıran aktör mimarisi üzerine kurulması kararlaştırılmıştır. Aktör geliştirimi sunan çerçeveler ile programlama dilleri göz önüne alındığında ve topluluk desteği (community support), belgeleme (documentation), programlama dili uygunluğu, nesneye dayalı geliştirim vb. özellikler hesaba katıldığında Akka'nın kullanılmasına karar verilmiştir. Bölüm-2.4'te aktör model, Akka ve küme parçalama mimarisi detaylı olarak incelenmiştir.

4.3.2 Birleşim Algoritmasının Belirlenmesi

Bölüm-2.5'te literatürde varolan en temel ve yaygın birleşim yöntemleri olan iç içe döngülü birleşim, bağlamalı birleşim, sırala-birleştir birleşim ve karma birleşim algoritmaları temel anlamda incelenmiş, olumlu ve olumsuz yönleri açıklanmıştır. Bunların arasından, iç içe döngülü ve bağlamalı birleşim algoritmalarında alt sorgular elde edilen sonuçlara göre yeniden yazılmakta ve sırayla işletilmektedir. Dolayısıyla, bu iki algoritmaya dayanan sorgu işlemcileri sorgunun yeniden yazımından da sorumlu hale gelmektedir. Ek olarak bu algoritmalar alt sorguları daha hızlı işletebilmek adına paralellikten ödün vermektedirler. Nested loop join algoritması sorguların yalnızca tek bir yerel depo üzerinde işletildiği sistemlerde (ilişkisel veritabanı sistemleri gibi) daha

[40] https://www.actor-framework.org
[41] https://github.com/untu/comedy
[42] https://learn.microsoft.com/en-us/dotnet/orleans
[43] https://www.erlang.org
[44] https://elixir-lang.org

uygundur. Diğer taraftan bind join algoritması ise çoklu depo sistemleri için uygun olmakla birlikte, ana sorgunun az sayıda alt sorgu içerdiği durumlarda daha etkilidir. Çünkü bu algoritmada paralellik arka planda olduğundan, bir ana sorgu çok fazla alt sorgudan oluştuğu durumda bu alt sorguların çalıştırılabilmesi için birbirlerinin sonucunu beklemesi gerekecektir.

Tüm bu durumlar göz önünde bulundurulduğunda tez kapsamında gerçekleştirilecek olan çoklu depo sisteminde paralellik ön plana çıktığından karma ve sırala-birleştir birleşim algoritmalarının kullanımı daha uygun gözükmektedir. Sırala-birleştir algoritmasının karma birleşim algoritmasına göre üstün yönü, sonuçlar sıralı olduğunda ortaya çıkmakta ve ek olarak bellek daha etkin kullanılmaktadır. Sonuçlar sorguya ek bir ifade eklenerek veri kaynağı tarafında sıralanarak elde edilmesi mümkündür, fakat bu durum veri kaynağının sıralama özelliğinin olması ve sıralamayı etkin yapabilmesi ile mümkün olabilmektedir. Bunun mümkün olmadığı durumlarda sıralama maliyeti sorgu işlemcisine düşeceğinden algoritmanın performansı etkilenerek önemli ölçüde azalacaktır. Dolayısıyla çoklu depo sistemlerinde veri kaynakları çok çeşitli ve farklı özelliklerde olabileceğinden tez kapsamında karma birleşim algoritmasının kullanılmasına karar verilmiştir. Daha önceden gerçekleştirimi yapılmış olan dağıtık sparql sorgu motorunun yapısında da karma birleşim algoritması kullanılmış ve oldukça iyi sonuçlar vermiştir.

Ana sorgu, ayrıştırıcı tarafından alt sorgulara ayrıldıktan ve bu alt sorgular da ilgili yerel depolar üzerinde çalıştırıldıktan sonra elde edilen alt sorgu sonuçlarının ana sorgu sonucunu oluşturmak üzere birleştirilmesi gerekmektedir. Buna göre dağıtık sparql sorgu motorunun yapısında bulunan ve çoklu depo sisteminde de kullanılan Karma Birleşim İcracı (HashJoinPerformer) birimi, elde edilen bu alt sorgu sonuçlarının birleştirilerek ana sorgu sonucunun oluşturulmasından sorumlu olacaktır. Bu birimin yapısında Şekil-2.11'te işleyişi görülen basit (simple) karma birleşim algoritmasının gelişmiş bir türevi olan GRACE (Kitsuregawa et al., 1983) karma birleşim algoritması kullanılmıştır. GRACE birleşim algoritması veritabanı sistemlerinin işletim motorları için optimize edilmiştir ve bulunuşu oldukça eskiye dayandığından

bellek, disk ve işlemci gibi kaynakların oldukça kısıtlı olduğu dönemlerde tasarlanmıştır. Dolayısıyla algoritmanın içsel dinamikleri belleğin çok kısıtlı olduğu, doğrudan işlemci çekirdekleri üzerinde parallelliğin sağlanarak işlemcilere atanmış önbellek ile disk arasında okuma/yazma işlemlerinin yoğun olarak yapıldığı duruma göre eniyileştirilmiştir. Günümüzdeki bellek, disk, işlemci ve iş parçacığı teknolojileri ile bu kısıtların büyük oranda ortadan kaldırıldığı düşünüldüğünde, algoritmada bazı değişiklikler yapılarak sistemin gereksinimlerine ve çoklu iş parçacığı mimarisine uygun olarak düzenlenmiş ve geliştirilmiştir. Sistemin sorgu birleşim işlemlerinde kullanılan yeniden düzenlenmiş GRACE hash join algoritması Şekil-4.1'te görülmektedir.

```
Algorithm ParallelJoinManager^Actor.performHashJoin()
{    //En büyük kova boyutunu belirle
     m = BUCKET_SIZE
     //Bekleyen sonuç sayısını ilklendir
     w = 0
     //R ilişkisini kullanarak ilk kova haritasını oluştur
     For each tuple r in R do
          put r in bucket no ( h(r.a) mod m ) in bucketMap_R
     end
     //Q ilişkisini kullanarak ikinci kova haritasını oluştur
     For each tuple q in Q do
          put q in bucket no ( h(q.b) mod m ) in bucketMap_Q
     end
     bucket_R = get first bucket from bucketMap_R
     bucket_Q = get first bucket from bucketMap_Q
     //Tüm kovalar için birleşim işlemini paralel yap
     while bucket_R != EOR and bucket_Q != EOQ
          mergerActor = create a new bucket merger actor
          mergeMsg = create a merge message using bucket_1 and bucket_2
          send mergeMsg to mergerActor
          w = w + 1
          bucket_R = next bucket in bucketMap_R after bucket_R
          bucket_Q = next bucket in bucketMap_Q after bucket_Q
     end
}
Algorithm HashJoinPerformer^Actor.mergeBuckets()
{    //İki kovanın ilişkilerini kullanarak karma haritalar oluştur
     for each tuple r in bucket_1
          add r in R_k no h(r.a) in hashMap_R
     end
     for each tuple q in bucket_2
          add q in Q_k no h(q.b) in hashMap_Q
     end
     //İkinci karma haritayı ilk karma haritanın anahtarlarını kullanarak araştır
     for each key k in hashMap_1
         if Q_k ∉ { } then
               put R_k o Q_k in the output relation
         end
     end
}
Algorithm ParallelJoinManager^Actor.handleSubJoinResult()
{    //Beklenen sonuç sayısını azalt
     w = w - 1
     //Alınan sonucu ana sonuca ekle
     FinalResult = FinalResult U SubResult
     if w = 0 do
        return FinalResult
     end
}
```

Şekil 4.1: Paralel GRACE Karma Birleşim (Hash Join) Algoritması

Buna göre öncelikle ParallelJoinManager aktör sınıfında paralel işleme özelliği düşünülerek birleşim yapılacak sonuçlar kovalara (buckets) bölünür. Kovalar eşit miktarda sonuç içerdiğinden kova boyutu sahip olunan bellek miktarına ve işlemci çekirdeği sayısına göre uygun şekilde belirlenmelidir. Çok büyük olursa bölümlendirme ve birleşim yapılacak kova sayısı azalır fakat kova boyutu artar, çok küçük olursa da kova boyutu azalır fakat bölümlendirme artacağından birleşim yapılacak kova sayısı artar.

Kova boyutu belirlendikten sonra sırasıyla R ve Q sonuçlarının her bir satırı için ilgili satırın birleşim bağlayıcısı değerinin karma kodu (hash code) bulunur. Daha sonra bu kodun kova boyutuna göre modu alınarak bulunan indeksteki kovaya ilgili sonuç satırı eklenir. Kovalar için üretilen R ve Q karma tabloları üzerinde aynı anda ilerlenerek her adımda sırayla R ve Q kovaları elde edilir. Bir karma birleşim aktörü yaratılarak R ve Q kovalarından oluşan bir mesaj bu aktöre yollanır ve beklenen sonuç sayısı artırılır. Tüm kovalar için bu adım tekrarlanır ve karma birleşim işlemlerinin aktörler tarafından paralel olarak yapılması sağlanır.

HashJoinPerformer aktörü sınıfında karma birleşim işlemi yapılmaktadır. Sırasıyla R ve Q kovaları içindeki tüm satırlar ilerletilerek her bir satırın (r ve q) birleşim bağlayıcısı değeri için karma kodu (hash code) elde edilir. Daha sonra her bir koda karşılık gelen satır ilgili karma tablosuna eklenerek sırasıyla R ve Q karma tabloları oluşturulur. Daha sonra R karma tablosunun anahtarları (k) üzerinden ilerlenir ve her bir anahtarın Q tablosundaki değerine bakılır. Eğer k anahtarının Q karma tablosundaki değerler karşılığı Q(k) bulunuyor ise bu değerler k anahtarının R karma tablosundaki değerler karşılığı R(k) ile birleştirilerek ilgili kova çifti için ana sonuca eklenir. Bu işlem tüm k anahtarları için işletilerek ilgili kovalar için nihai sonuç elde edilir.

Son adımda ParalelJoinManager aktörü her bir kova çifti için üretilen alt sonuçları elde ettikçe bunu ana sonuca ekler ve beklenen sonuç sayısını azaltır. Beklenen sonuç sayısı sıfıra ulaştığında ana sonuç elde edilmiş demektir.

Algoritma; kova tablosu oluşturma, karma tablosu oluşturma ve doğrulama aşamaları bakımından Grace karma birleşim algoritmasıyla benzerdir. Fakat

oluşturulan kovalar Grace algoritmasında sırayla işletilmektedir. Halbuki Şekil-4.1'de geliştirilen algoritmada kovalar paralel bir şekilde aktörler tarafından birleştirilmekte ve bu da birleşim maliyetini önemli ölçüde azaltmaktadır. Tez kapsamında geliştirilecek çoklu depo sisteminin sorgu işleme altyapısının da paralel, hızlı ve ölçeklenebilir olması gerektiğinden, birleşim mekanizması da buna uygun şekilde tasarlanmıştır. Ayrıca iç içe döngülü birleşim ve bağlamalı birleşimde olduğu gibi sorgu yeniden yazmanın olmaması, önemli bir geliştirim maliyetinin önüne geçmektedir. Ek olarak sırala-birleştir birleşim yönteminde de sorgu sonuçlarının veri deposu tarafında sıralanması sorgu işletim maliyetini düşürmek açısından önem taşıdığından bu durum veri deposunun sorgu işleme yeteneğine bağlı olmayı gerektirmektedir. Dolayısıyla bahsi geçen algoritmaların tüm kısıtları ve yetenekleri göz önünde bulundurulduğunda karma birleşim algoritması büyük veri kümeleri ile çalışabilme, paralelleştirilebilme, sorgu yeniden yazma bağımsızlığı ve veri deposu sorgu yeteneği bağımsızlığı, birçok durumda hızlı ve etkin çalışabilme gibi çok farklı avantajlara sahip olduğundan paralel ve ölçeklenebilir sorgu işleme yöntemleriyle en uyumlu çalışabilen çözüm olarak karşımıza çıkmaktadır.

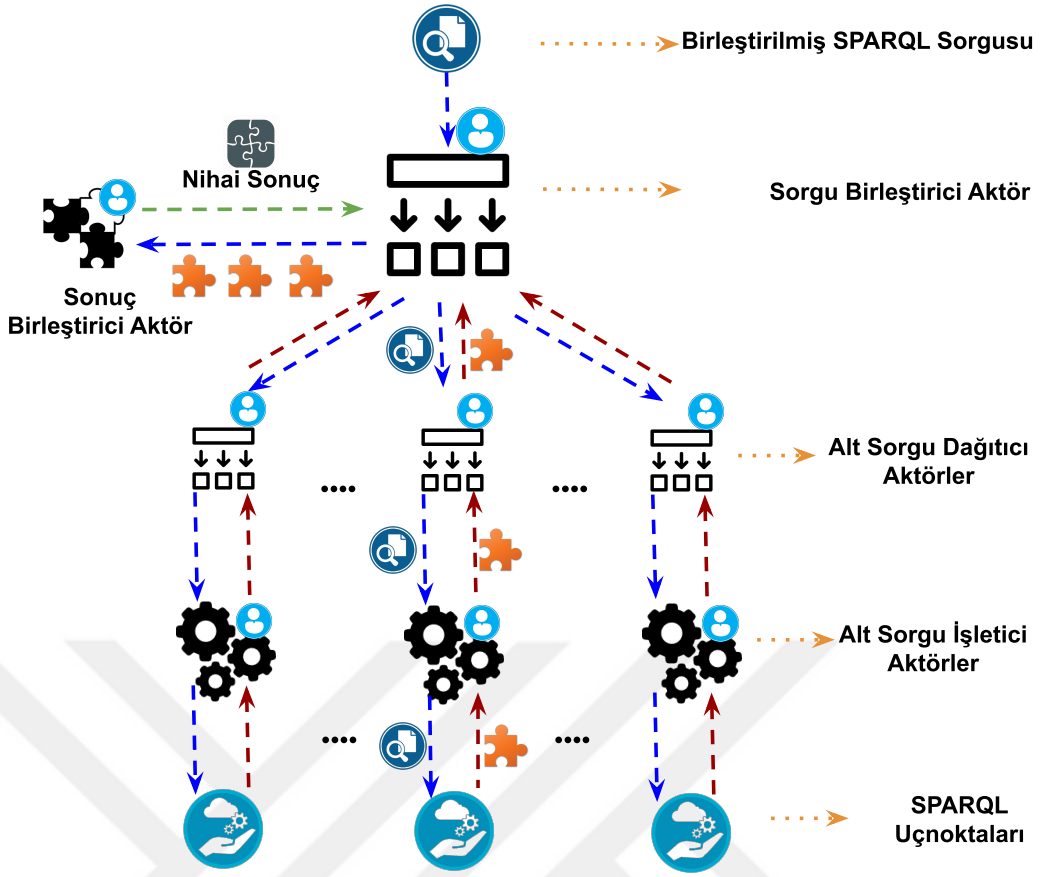
4.3.3 Aktör Tabanlı SPARQL Sorgulama Mimarisi

Çoklu depo sisteminde ihtiyaç duyulan sorgu motorunun sorgu dağıtımı ve sorgu işletimi noktalarında paralelliği ve ölçeklenebilirliği sağlaması gerekmektedir. Bu durumda her sorguyu çözümleyen ve işleyen yapılar birbirinden bağımsız çalışır nitelikte olmalıdır. Bu açıdan aktör tabanlı mimari, hem çalışma hem de haberleşme ilkeleri bakımından ölçeklenebilir sorgu işleme altyapısının oluşturulmasına olanak sağlamaktadır. Bu mimaride iş gerçekleştiren birimler re-aktif yapıda çalışan aktörlerdir ve birbirleriyle mesajlaşma yoluyla iletişime geçerler.

Yüksek lisans tezinde SPARQL sorgulama ve birleşim eniyileştirmeleri (join optimizations) üzerine çalışmalar yürütülmüştür. Bu çalışmaların doktora öğrenimi boyunca devamında ise SPARQL sorgu sonuçlarının karma birleşimi kapsamında bir modül geliştirimi yapılmıştır. Daha sonra bu modül yüksek

lisans öğrenimi boyunca çalışmış olduğum "Etmenler için Anlamsal Ortam Tasarımı ve Gerçekleştirimi" isimli TÜBİTAK projesinde, sorgu takibi yapan etmenlerin takip ettikleri alt sorguların sonuçlarının değişmesi durumunda ana sorgu sonucunun oluşturulması adımında kullanılmıştır. Yüksek lisans süresince yürütülen tez ve TÜBİTAK projesinde bahsedilen çalışmaların harmanlanması sonucu araştırma projesi olarak doktora ders dönemi içerisinde Anlamsal Veri Bütünleştirme dersi kapsamında aktör tabanlı ve temelinde Şekil-4.1'deki paralel karma GRACE birleşim algoritmasının küme üzerinde çalışma özelliği bulunmayan ilk versiyonunu barındıran AQuA (Actor based sparql Querying Architecture - Aktör tabanlı sparql sorgulama mimarisi) isimli bir sorgu motoru geliştirilmiştir. Şekil-4.2'de çalışma prensibi görülmekte olan, Akka aracını kullanan ve Java dilinde yazılmış AQuA çalışma prensibi olarak kendisine gelen birleştirilmiş (federated) SPARQL sorgularını ayrıştırıp alt sorgulara ayırmakta ve ayrıştırılan bu alt sorguları ilişkili oldukları uç-noktalarda (endpoint) çalıştırarak dönen sonuçları birleştirip ana sonucu oluşturmaktadır. Bu aracı geliştirmenin temel sebebi geliştirilen birleşim algoritmasının etkinliğinin ortaya konulması idi. Dolayısıyla geliştirilen bu aktör tabanlı SPARQL sorgu motoru, yüksek lisans tezi kapsamında bağlamalı birleşim ve önbellekleme kullanılarak eniyileştirilen WoDQA (Akar et al., 2012) aracı ile sorgu işletimi bakımından kıyaslanarak değerlendirilmiştir. Kıyaslamada FedBench[45] (Schmidt et al., 2011) kalite testinin (benchmark) 7 adet çapraz alan (cross domain) ve 7 adet yaşam bilimleri (life sciences) sorgusu kullanılmıştır. Elde edilen sonuçlara göre tüm sorgular zaman aşımı olmadan çalıştırılmıştır. Değerlendirme sonuçları Tablo-4.1'de görülmektedir.

[45]https://code.google.com/archive/p/fbench



Şekil 4.2: AQuA Mimarisi ve Sorgu İşleme Akışı

Bu sonuçlara göre sorgu işletiminin tüm sorgular için makul zaman aralığında gerçekleştiği söylenebilecektir. Tablodan görüldüğü üzere sorgu işletimi neredeyse tüm sorgular için tutarlı ve benzer süreler içerisinde tamamlanmaktadır. Ancak yaşam bilimleri alanındaki ikinci sorgu dışındaki tüm sorgular WoDQA'ya göre daha yavaş çalıştırılabilmiştir. Bunun sebebi FedBench'teki bu çapraz alan ve yaşam bilimi sorgularının çoğunun içerdiği en az bir adet seçici alt sorgu bulunmasıdır. WoDQA hem bir sorgu çözümleyicisi hem de sorgu motoru olduğundan, alt sorgu sıralaması yaparak sorgu eniyileştirmesi gerçekleştirmektedir. Bu durumda seçici alt sorguyu önce işleterek, bağlamalı birleşim yaparken bir sonraki işletilecek alt sorguya "FILTER" bloğu içinde daha az sonuç bağlayarak hem sorgunun seçiciliğini artırır, hem de daha hızlı filtrelenmesini sağlar. Geliştirilen AQuA sorgu motoru ise sorgu analizi işlemi yapmayıp yalnızca verilen birleştirilmiş (federated) SPARQL

Sorgu Adı	WoDQA (ms)	AQuA (ms)
Çapraz Alan-1	16	49
Çapraz Alan-2	11	35
Çapraz Alan-3	18	691
Çapraz Alan-4	10	617
Çapraz Alan-5	11	606
Çapraz Alan-6	85	786
Çapraz Alan-7	15	730
Yaşam Bilimleri-1	14	162
Yaşam Bilimleri-2	4198	214
Yaşam Bilimleri-3	303	1922
Yaşam Bilimleri-4	10	757
Yaşam Bilimleri-5	118	1649
Yaşam Bilimleri-6	12	711
Yaşam Bilimleri-7	44	635

Tablo 4.1: AQuA aracının FedBench Sorgularının İşletimi üzerinden WoDQA ile kıyaslanması

sorgusunu işletir. Bunu yaparken de tüm alt sorguların kendi başına işletilmesi gerektiğinden, kısmen maliyetli alt sorguların işletimi süre alabilir. Ancak alt sorgu işletimi bittikten sonraki birleşim işlemi oldukça hızlı yapılmaktadır. Yaşam bilimleri-2 (Life Sciences-2) sorgusunda olduğu gibi tüm alt sorguların seçiciliği düşük olduğunda veya filtreleme işlemi etkin yapılamadığında WoDQA'nın sorgu işletim performansı düşmektedir. Ancak karma birleşimde filtreleme gibi veri kaynağı tarafındaki işlevselliklere gerek duyulmadığından birleşim işlemi oldukça hızlı bir şekilde yapılıp tamamlanabilmektedir. Dolayısıyla büyük ve heterojen veri kaynaklarının sorgulanması kısmında algoritmanın etkinliği ön plana çıkmakta olup, homojen veri kaynaklarının sorgulanmasında da oldukça etkili ve rekabetçi olduğu söylenebilecektir.

4.3.4 AQuA'ya Sorgu Takibi Özelliğinin Kazandırılması

Bir önceki bölümde anlatılan AQuA aracında aktör yaratımı için Akka uzaklama (remoting) yöntemi kullanılmıştır. Buna göre bir bilgisayar düğümündeki JVM üzerinde çalışan bir aktör sistemi, farklı ve uzak bir bilgisayar düğümündeki başka bir JVM üzerinde çalışan aktör sistemi ile bağlantı kurarak aktörler yaratabilmekte ve yaratmış olduğu bu aktörlere mesaj yollayabilmektedir. Bu sayede bir bilgisayar düğümü üzerinde çalışan aktör sistemi farklı bilgisayar düğümleri üzerinde dağıtılıp koşuldukça, uzaklama ile birbirleri üzerinde aktörler yaratabilmektedirler. İşte AQuA'da da bu mantık kullanılarak belirlenen bir ana düğüm diğer düğümlerin kaynaklarını kullanarak tıkanıklık durumlarında uzak düğümler üzerinde aktörler yaratılabilmiş ve sistem çökmeden işleyişine devam edebilmiş, böylece de ilk versiyon ölçeklenebilirlik özelliği sağlanmıştır. İlerleyen bölümde öncelikle AQuA'ya sorgu takibi özelliği kazandırılması ve daha sonra da aracın ölçeklenebilirliğinin iyileştirilerek yaratılan sorgu takip ortamı mimarisinden bahsedilecektir.

Yüksek lisans döneminin sonu ve doktora döneminin başındaki dönem içerisinde, yüksek lisans öğrenimi sırasında çalışılan "Etmenler için bir Anlamsal Ortam Tasarımı ve Gerçekleştirimi" isimli TÜBİTAK projesindeki çalışmalar ışığında, AQuA aracının temelinde de kullanılan karma birleşim

modülü ve CArtAgO[46] (Ricci et al., 2009) aracı kullanılarak etmen tabanlı bir "sorgu takip ortamı"nın gerçekleştirilmesine katkıda bulunulmuştur. Bahsedilen sorgu takip ortamının gerçekleştirilmesini takiben yapılan yayın çalışması Journal of Web Semantics[47] dergisine sunulmuştur ve çalışmada ortaya konan değerlendirme sonuçlarına göre sistem belirli bir bileşen (artifact: CArtAgO'da ortaya atılan etmenler tarafından kullanılan ortam bileşenleri), etmen ve sorgu sayısına ulaştığında kullanılmaya uygun serbest sistem kaynakları bulunmasına rağmen sorgu takip edemez hâle gelmektedir. Dolayısıyla sistemin ölçeklenebilir olmamasından dolayı bu çalışma reddedilmiştir. Buradan hareketle doktora öğrenimi içerisinde 2016 yılının başından başlanarak ilk tez izleme dönemini de kapsayacak şekilde aktör tabanlı ölçeklenebilir bir sorgu takip ortamı projesi üzerinde çalışılmıştır. Bu projenin temelini de Bölüm-4.3.3'de detaylandırılan AQuA sorgu motoru oluşturmaktadır. Ancak bu sorgu motorunun ilk geliştirildiği anda ölçeklenebilir özellikte olmamasından dolayı Bölüm-4.3.4'ün girişinde bahsedildiği gibi Akka uzaklama (remoting) özelliği kullanılarak sorgu motoruna ölçeklenebilirlik ve dergiye sunulan çalışmadakine benzer olarak sorguları takip edebilme yeteneği kazandırılmıştır.

Buna göre sorgu takip ortamının mimarisini anlatmadan önce "ortam"ın ne anlam ifade ettiğini açıklamak anlaşılabilirlik açısından uygun olacaktır. Ortam (environment); belirli bir kapalı dünya kabullenmesini ve burada yer alan tüm varlıkları ve eylemleri ifade etmektedir. Kapalı dünya kabullenmesi de aslında bir kullanım durumu olarak ifade edilebilecektir. Sorgu takibini gerektiren bir kullanım durumu ortaya atılmış ve böylece hem sistemin işleyişinin modellenmesi hem de sistemin tasarlanması kolaylaşmıştır.

Sorgu takip ortamı, kendisine gelen sorgu takibi isteklerine öncelikle sorguyu işletip sonucunu bildirerek cevap verir. Daha sonra bu sorguda değişiklik olup olmadığını belirlemek üzere, sorguyu belirli aralıklara sorgunun işletildiği veri kümeleri üzerinde çalıştırarak sonuçları kontrol eder. Eğer yeni bir değişiklik varsa yeni sorgu sonucunu kendisinden sorgu takibini

[46]http://cartago.sourceforge.net

[47]https://www.sciencedirect.com/journal/journal-of-web-semantics

talep eden varlığa (aktör, etmen, nesne vb.) bildirir. Ortamda takip edilecek sorguların tipi o zamanki gereksinimden ötürü SPARQL olarak belirlenmiştir. Ayrıca veriler de RDF depolarına yüklenerek SPARQL uçnoktası aracılığıyla sorgulamaya açılmıştır. Sorgu takip ortamındaki sorgu işleme mimarisi AQuA aracının sorgu takibi özelliği eklenmiş hâli olup Şekil-4.2'de görülmektedir.

Ölçeklenebilir aktör tabanlı sorgu izleme mimarisinin önceki gerçekleştirimlerinde, ölçeklenebilirlik ayrı bir ara katman yazılımı uygulanarak ele alınmaya çalışılmıştı. Ara katman (middleware) yazılımı, Akka'nın uzaklama özelliğini kullanarak yeni aktörleri ayrık (discrete) düğümlere yerleştirmekten sorumluydu ve bir düğümün aktör sınırına ulaşıp ulaşmadığını kontrol ediyordu. Akka uzaklama farklı makineler üzerinde çalışan sistemlere erişip, buralarda aktörler yaratmaya ve bu aktörlere mesaj yollamaya olanak sağlamaktadır. Buna göre öncelikle sistem birden fazla makine üzerinde çalıştırılmaktadır. Makinelerden biri yönetici (manager) olarak, diğerleri de işçi (worker) makineler olarak seçilmektedir. Yönetici makinenin konfigürasyonunda, her makine için belirlenen maksimum aktör sayısı parametresi yer almaktadır. Bir makinede maksimum sayıya ulaşıldığında, yönetici makine müsait olan bir sonraki makine üzerinde aktör yaratmaya başlar. Buna göre aktör sayısı sabit olarak alınmıştır. Ara katman, aktörlerin düğüm sayılarını ve aktör referanslarını Redis deposunda saklamaktaydı.

Buradaki problemlerden biri ölçeklenebilirliğin sistem yükü miktarı veya eşit dağılım gibi akıllı yöntemlere göre değil de, statik belirlenen bir sayıya göre yapılmasıdır. Bir diğer problem ise aktör dağıtım (deployment) özelliğinin ve dolayısıyla da sistem ölçeklenebilirliğinin tek bir yönetici üzerinden sabit kodlanmış (hard-coded) yolla yönetilmesidir. Dolayısıyla aktör dağıtımı için yönetim ve aktörler için de yer takip maliyeti ortaya çıkmaktadır. Bu durum yönetim yükünün tek bir bileşene kaydırılması sonucunu doğurarak bir darboğaza neden oldu. Ayrıca, ne düğümlerin kaynak kullanımını izleyen ne de bir dağıtım algoritması kullanan, yalnızca statik bir aktör sayısıyla yönetilen ilkel seviyedeki bir ölçeklenebilirlik yaklaşımı, aktör modelinin tam kapasitesini ve esnekliğini kısıtlamaktaydı. Bu ne-

denle, daha sonra yapılan araştırmalar sonucu Akka'nın aktörler için konum şeffaflığı sağlayan, dağıtımlarını ve konuşlandırmalarını kendisi yöneten küme parçalama özelliğinin kullanılmasına karar verilmiştir.

Sistemin öncelikli gereksinimi, ölçeklenebilirliğin otomatik olarak ve akıllı bir yöntemle yapılmasıdır. Bunun için Akka küme parçalama özelliğinin kullanımına geçilmiştir. Akka kümesi, merkezi olmayan eşler arası (peer-to-peer) ve hataya toleranslı bir servis sunarak, sistemi darboğaz (bottleneck) ve hata durumlarından korur. Bunu da dedikodu protokolü (gossip protocol) ve otomatik bir hata bulucu (failure detector) ile sağlar. Akka kümesinde aktör sistemlerini içerecek makineler kümeye düğüm olarak eklenmektedir. Başlangıç düğümleri tanımlandıktan sonra, bu düğümler üzerinde çalışan aktör sistemleri kendilerini kümeye kaydettirerek katılmaktadırlar. Küme için yönetici, dedikodu yakınsaması (gossip convergence) yöntemiyle seçilmektedir. Yönetici düğüm ise küme yakınsamasını ve üyelik durumu geçişlerini yönetir. AQuA sorgu motorunun Akka küme parçalama yöntemiyle dönüştürülerek sorgu takibi özelliği eklenmesi sonucu gerçek anlamda ölçeklenebilir bir yapıya sahip olan MonARCh (Monitoring Architecture for SPARQL Result Changes - SPARQL sorgu değişiklikleri için bir izleme mimarisi) aracı ortaya çıkmıştır.

4.3.5 MonARCh: Sparql Sorgu Sonucu Değişikliği İzleme Mimarisi

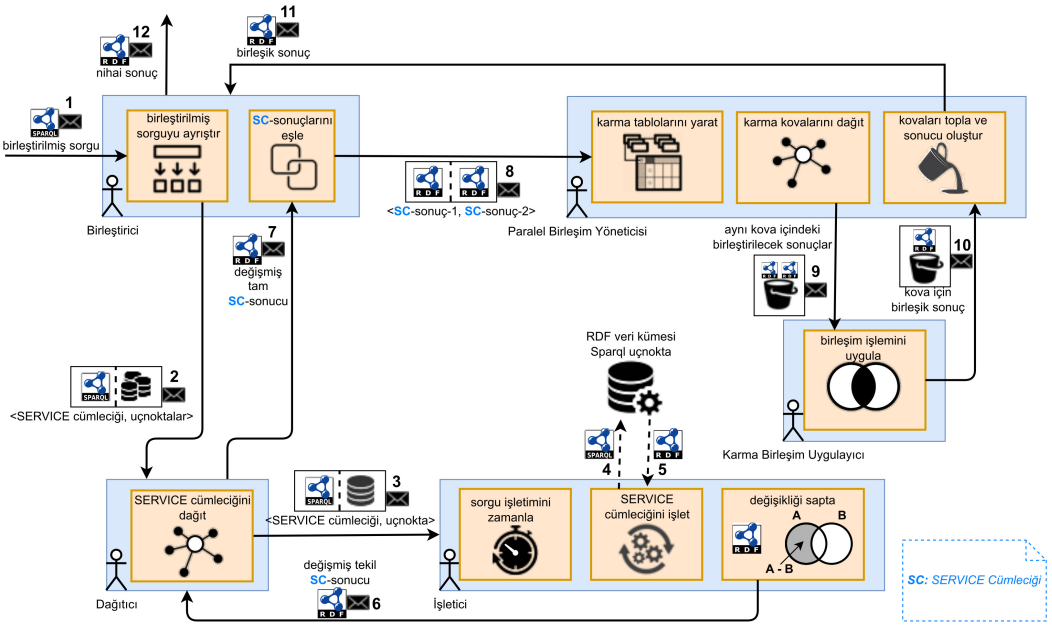
MonARCh, Bağlı Veri ortamında birleşik SPARQL sorguları için bir izleme mimarisidir. İzleme istekleri çok sayıda eşzamansız uygulamadan gelebileceğinden, böyle bir mimari tasarlanırken ölçeklenebilirlik kritik bir husus olmaktadır. Aktör model, ölçeklenebilirlik gerektiren eşzamanlı hesaplama sorunları için iyi tasarlanmış bir çözümdür. Bu nedenle MonARCh, aktör modelinin üzerine inşa edilmiştir.

Literatürdeki izleme altyapıları genellikle çekme (pull) veya itme (push) yaklaşımına dayalıdır. Çekme yaklaşımında tüketici, veri kümelerini periyodik sorgular aracılığıyla izler. İtme mekanizmasında ise tüketici, veri yayıncısına abone olur ve ardından veri yayıncısı, abonelerdeki değişiklikleri kabul eder. MonARCh, her iki yaklaşımı birleştirir ve birleşik (combined) bir itme-çekme

(pull-push) mekanizması kullanır. Çekme işlemi, Veri Webi'nin güncellemeler için sorgulanmasını tanımlarken, itme işlemi, SPARQL sonuç değişikliklerinin kayıtlı istemcilere (clients) bildirilmesi anlamına gelir. Çekme yaklaşımı kullanılarak, tüm yeni veri kümeleri, sorgulamadan önce her seferinde dinamik olarak keşfedilir. Veri Webi'nin belirli bir bölümünü izlemesi gereken uygulamalar, isteklerini SPARQL sorguları aracılığıyla izleme hizmetine kaydeder. Bir sonuç kümesi değişikliği durumunda bilgilendirilirler. Bu birleşik itme-çekme yaklaşımını kullanmak, MonARCh'ın özgün yanlarından birini oluşturur.

Önerilen izleme mimarisinin diğer özgün yanı, birleştirilmiş (federated) bir sorguyu SERVICE cümleciklerine (SERVICE clause) ayırma ve her birini bağımsız olarak çalıştırma yeteneğidir. Bu SERVICE cümlecikleri, ilgili veri kümelerinin değişiklik frekanslarına göre çalıştırılır ve bu amaçla DaDy üst verilerinden yararlanılır. DaDy üst verileri, Bağlı Veri kaynaklarının VoID belgelerinde bulunur ve bu kaynakların değişim frekanslarını tanımlar. Kısmi (partial) sonuçlar, yeni işletilen bir SERVICE cümleciğinden veya daha önceden işletilen SERVICE cümleciklerinin yerel kısmi sonuç önbelleklerinden gelebilir. Bu kısmi sonuçlar daha sonra paralel GRACE karma birleşim algoritması kullanılarak birleştirilir. Böylece, her bir SERVICE cümleciğinin ayrı ayrı izlenmesi, sistemin belirli bir SERVICE cümleciğinin önbelleğe alınmış kısmi sonucunu yeniden kullanmasına olanak tanır. Bu önbelleğe alınmış sonuç kümeleri, bunları içeren birleştirilmiş sorguların nihai sonucunu elde etmek için bir araya getirilebilir.

Özetlemek gerekirse, MonARCh'ın öne çıkan ve ayırt edici özellikleri, aktör tabanlı ölçeklenebilirlik, mimarinin temelinde birleşik bir itme-çekme yaklaşımı ve veri kümesi değişiklik frekanslarına dayalı olarak SERVICE cümleciklerini bağımsız olarak izleme yeteneğidir. MonARCh, (Wieringa, 2014) tarafından tanımlanan tasarım bilimi metodolojisine göre geliştirilmiş ve değerlendirilmiştir. Bu metodolojiye uygun olarak, türetilen DBpedia, New York Times ve borsa veri kümelerinin kullanıldığı kapsamlı bir durum çalışması geliştirilmiştir. Sistemin değerlendirilmesi için ortaya konan durum çalışmasından faydalanılarak farklı özelliklerde, sayıda ve seçicilikteki üçlü de-



Şekil 4.3: MonARCh'ın Sorgu İzleme Akışı

senlerinden oluşan sorgu kümeleri tasarlanmıştır. Ayrıca sonuç filtreleme özelliği ile de seçicilik, sorgu tipi özelinde de derecelendirilmiştir. Sistemin sorgu performansı sorgu işleme süreleri ile, ölçeklenebilirlik performansı ise kullanılan CPU, bellek ve ağ üzerinden iletilen mesaj boyutu metrikleri ile ölçülmüştür. Yalnızca değerlendirme sonuçlarının toplanması dahi ileri-letme (iteration) sayısının fazlalığından ötürü 3 ay kadar sürmüştür. Böylece sorgu performansının ve ölçeklenebilirlik sınırlarının sınandığı kapsamlı bir değerlendirme ile sistemin etkinliği ve başarısı ortaya konmuştur.

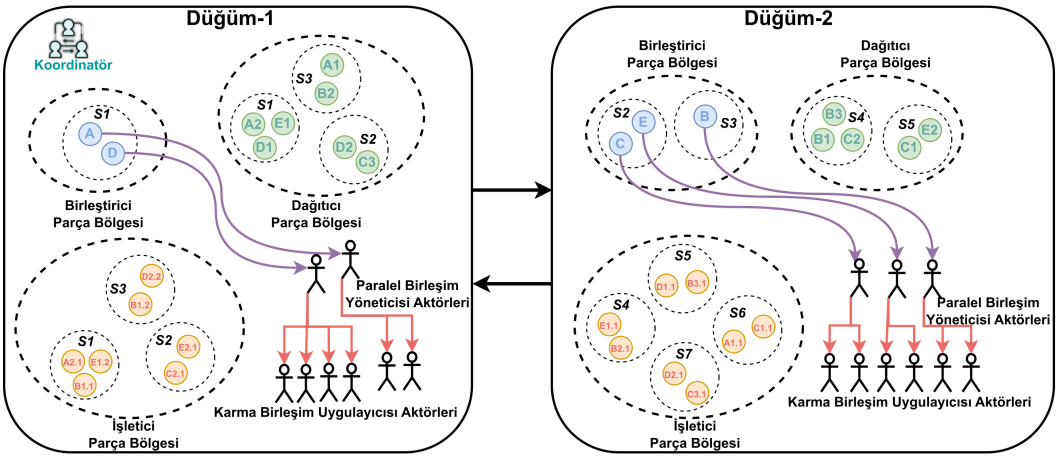
4.3.5.1 Soyut Mimari Tasarımı

MonARCh'ın soyut mimari tasarımı, deneysel uygulaması ve en önemlisi sistematik bir değerlendirmesi için Tasarım Bilimi Metodolojisi (Wieringa, 2014) ilkeleri uygulanmıştır.

MonARCh'ın soyut mimarisi Şekil-4.3'te tasvir edilmiştir. Temel olarak; Birleştirici (Federator), Dağıtıcı (Distributor), İşletici (Executor), Paralel Birleşim Yöneticisi (Parallel Join Manager) ve Karma Birleşim Uygulayıcı (Hash Join Performer) olmak üzere beş bileşenden oluşmaktadır. Birleştirici, Dağıtıcı ve İşletici, sorgu izleme akışını yönettiğinden kalıcı olması ve her

birinin küme parçalama sistemi içerisinde kendi özel parça bölgesi (shard region) olması gereklidir. Bu bölgeler, uygun aktör örneklerinin oluşturulmasını ve aralarındaki mesaj yönlendirmesini kontrol eder. Diğer yandan Paralel Birleşim Yöneticisi ve Karma Birleşim Uygulayıcı ise esasen sonuç odaklıdır. Sonuçları farklı olabilmekte ve zaman içinde değişebilmektedir. Sonuç olarak, bu bileşenlerin her bir birleştirme işlemi için yeniden oluşturulması ve kalıcı olmaması gerekir.

Eğer SPARQL sorgusu birleştirilmiş (federated) biçimdeyse, Birleştirici sorguyu SERVICE cümleciklerine böler. Algılanan her SERVICE cümleciği için ilgili SPARQL uç noktaları atanır. <SERVICE cümleciği, uç noktalar> çiftleri daha sonra Dağıtıcı bölgesine gönderilir ve bu bölge mesajları uygun Dağıtıcı aktörlerine iletir (Şekil-4.3 adım 2). Birleştirici, Dağıtıcılar tarafından teslim edilecek SERVICE cümlecikleri için sonuçların takibinden de sorumludur. Çünkü bu SERVICE cümleciği sonuçları daha sonra Paralel Birleşim Yöneticisine gönderilerek tek bir ana sonuçta birleştirilir (Şekil-4.3'teki 8. ve 9. adımlar). Benzer şekilde, Dağıtıcı <SERVICE cümleciği, uç noktalar> çiftine sahip bir mesaj aldığında, İşletici bölge aracılığıyla her uç nokta için ilgili İşletici aktöre SERVICE cümleciğini gönderir (Şekil-4.3'teki 3. adım), sonra sonuçları toplar ve birleştirir. Bir SERVICE cümleciğinin işletileceği SPARQL uç noktası çalışır durumda değilse, İşletici önbelleğe alınan son sonucu döndürür. Son olarak, İşletici bir <SERVICE cümleciği, uç nokta> çifti içeren bir mesaj aldığında, SERVICE cümleciğini ilgili uçnokta üzerinde işletir. Bu modül, bir SERVICE cümleciğinin sonucundaki değişiklikleri algılamaya çalışan anahtar bileşendir. Sorguyu yeniden işletmek ve bitiş noktasının tahmini değişiklik aralığına dayalı olarak yeni sonuçları elde etmek için kendi kendini zamanlar (schedule). Herhangi bir değişiklik olup olmadığını algılamak için eski sonuçlar yeni sonuçlar sonuçlar ile karşılaştırılır. Buna göre bir değişiklik tespit edildiğinde Dağıtıcı'ya iletilir. Dağıtıcı, birden çok uçnokta üzerinde işletilebilen ilgili SERVICE cümleciğinin yeni sonuçlarını toplar ve bu SERVICE cümleciği için birbirine eklenmiş sonuçları tek bir sonuç olarak Birleştirici'ye yollar.



Şekil 4.4: MonARCh için Örnek Küme Yapısı Modeli

Birleştirici yeni (değişmiş) bir sonuç aldığında, ortak sorgu değişkenini kullanarak eşleşen bir SERVICE cümleciği sonucu olup olmadığını kontrol eder. Daha sonra Paralel Birleşim Yöneticisi'ne birleşim planı için iki SERVICE cümleciği sonucu gönderilir. Paralel Birleşim Yöneticisi, belirlenen sabit kova (bucket) boyutuna göre Paralel Grace Karma Birleşim algoritmasını kullanarak her iki sonuç için de hash tabloları oluşturur. Karma tablo (hashmap) kovaları, birleşim işleminin paralel yürütülmesi için anahtarlara (keys) göre eşleştirilir ve her bir çift, ayrı bir Karma Birleşim Uygulayıcı aktöre gönderilir. Karma Birleşim Uygulayıcı, basit bir karma birleşim işlemi gerçekleştirir ve birleştirilen sonuçları Paralel Birleşim Yöneticisi'ne döndürür.

Paralel Birleşim Yöneticisi, tüm kovalardan birleştirilmiş sonuçları toplar, birbirine ekler ve nihai birleştirilmiş sonucu üretir. Birleştirici, nihai sonucu ilk kez veya değişiklik olarak elde ettiğinde, bunu ana birleştirilmiş SPARQL sorgusunu yollayan temsilciye iletir.

4.3.5.2 Soyut Mimari Gerçekleştirimi

İzleme mimarisini oluşturmak ve kümeyi yönetmek için, prototip sistemde JVM üzerindeki (Java Virtual Machine) aktör model gerçekleştirimi olarak Akka[48] araç seti (Gupta, 2012) kullanılmıştır. Yerleşik eşzamanlılık

[48] `https://akka.io`

yönetimine (built-in concurrency management) sahip olduğu ve aynı zamanda Akka'nın gerçekleştirim dili olduğu için, programlama dili olarak ise Scala[49] seçilmiştir. Akka'nın küme parçalama özelliği sayesinde sistem yüksek düzeyde ölçeklenebilir ve eşzamanlıdır. Ayrıca, yüksek performansa sahip olup, CPU ve belleğe bağlı binlerce aktöre, bunun yanında bilgisayar kümesi düğümü başına on binlerce msg/sn'ye ulaşabilen mesaj iş yüklerinin patlamasına (burst of message workloads) karşı dirençlidir.

Şekil-4.4, Akka küme parçalama özelliği ile oluşturulmuş örnek bir MonARCh kümesini gösterir. Helland'ın (Helland, 2007) çalışması, Akka'nın küme parçalama mimarisinin köklerine ilham vermiştir. Mesajlar, bu mimaride aktörler arasında sorgu sonuçları göndermek için kullanılır. Bazı sonuçlar çok büyük olabileceğinden, farklı düğümlerde çalışan aktörler arasında serileştirme ve TCP iletişimi yoluyla iletim zor olabilir. Sonuç olarak, zamana duyarlı uygulamalar için daha uygun olan ve daha büyük mesajları kolayca iletebilen UDP mesajlaşma yapılandırması kullanılmıştır. Küme, dedikodu protokolünü kullanarak gelen mesajları uygun düğüme yönlendiren bir koordinatör aktöre sahiptir. Uygulanan sistemin üç kritik modülü, Birleştirici, Dağıtıcı ve İşletici aktör bölgeleridir. Bu modüller ebeveyn-çocuk (parent-child) ilişkilerine sahiptir ve varlık aktörlerinin oluşturulmasını ve mesaj yönlendirmeyi kontrol ederler. Parçalar, basitçe, bölgelerde bulunan varlıklar olarak adlandırılan aktör örnekleri gruplarıdır. Böylece bu yapı, gerektiğinde bu varlıkların daha sonra taşınmasını kolaylaştırır. Daha önce bahsedilen küme mimarisi, aktörler için konum şeffaflığı sağladığından, hangi aktörün hangi düğümde çalıştığını bilmek gereksizdir. Mesajlar basitçe herhangi bir düğümün bölgesine gönderilir ve aktör dağıtımı ve mesaj teslimi için yük dengeleme de dahil olmak üzere geri kalanı küme yönetimi halleder. Varlık aktörleri, S harfi ve parça numarası ile kodlanmış parçalar halinde düzenlenmiş renkli dairelerle harfler (bazıları sayılarla) olarak tasvir edilir. Her küme düğümünün kendi parça bölgesi vardır ve parçalar, aynı mantıksal bölge temelinde küme genelinde dağıtılır. Birleştirici, Dağıtıcu ve İşletici adlı üç parça bölgesi vardır. Bu bölgelerin

[49] `https://www.scala-lang.org`

sahibi her iki düğümdür ancak Şekil-4.4 içinde gösterilen küme olarak yönetilir. Örneğin, B harfiyle kodlanan ve mavi bir daire olarak temsil edilen birleşik bir sorgu, Düğüm-2'nin Birleştirici parça bölgesinde bulunur. İlgili uçnoktalara sahip SERVICE cümlecikleri, Dağıtıcı parça bölgesinde yeşil daireler olarak temsil edilen B1, B2 ve B3 ile aynı harf ve sorgu numarasına sahiptir. Varlıklar mantıksal olarak aynı bölgede yer almaktayken, farklı düğümlerde de yer alabilmektedirler. (B2, Düğüm-1'de, B1 ve B3 ise Düğüm-2'de). İşletici parça bölgesinde, ilgili uç noktalara atanmış SERVICE cümlecikleri, "SERVICE cümlecik kodu", bir "nokta" ve bir "uç nokta numarası" şekilde B1.1, B1.2, B2.1 ve B3.1 olarak kodlanır. Paralel Birleşim Yöneticisi ve Karma Birleşim Uygulayıcı aktörleri bir küme tarafından yönetilmediğinden, üst aktörleri ile aynı düğümde bulunurlar. Her ikisi de yalnızca ilgili sonuçları birleştirir ve belirli iki sonuç için bir birleşim işlemi yalnızca bir sefer yürütülür ve birleştirilmiş sonuç yeniden kullanılmaz. Bu nedenle, birleştirme işlemi tamamlandıktan sonra bu aktörler yok edilir. Birleştirici, Paralel Birleşim Yöneticisi'ni ve Paralel Birleşim Yöneticisi aktörleri de Karma Birleşim Uygulayıcısı'nı yaratır. MonARCh'ın kaynak koduna aşağıdaki URL üzerinden erişilebilmektedir: *https://github.com/seagent/monarch*.

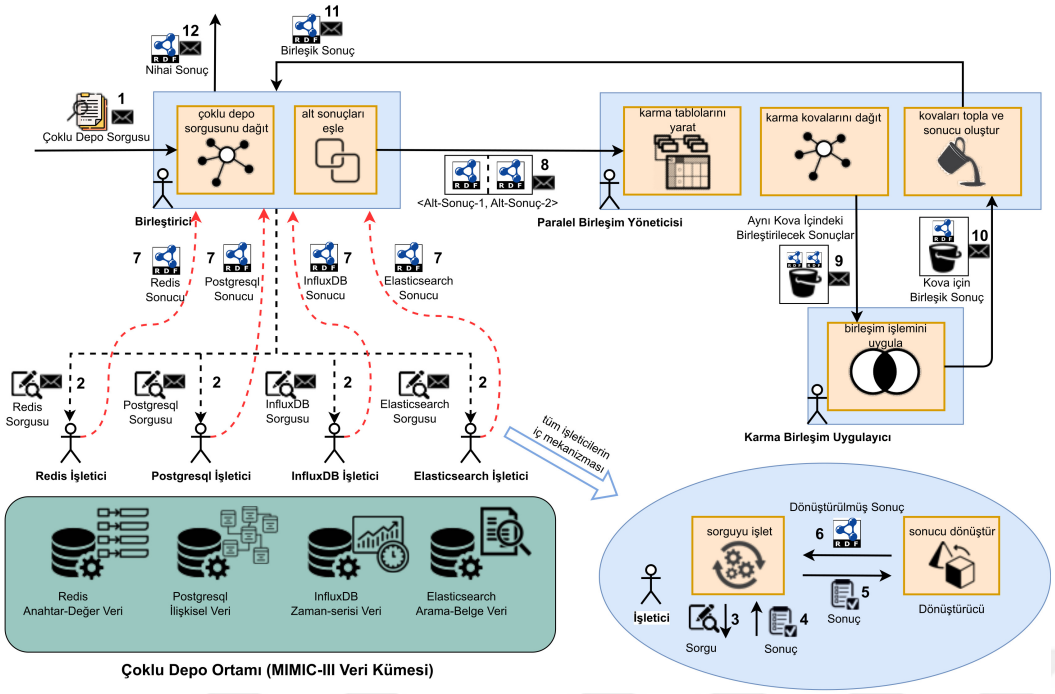
4.3.6 Sorgu İşletim Motorunun Çoklu Depo Sistemine Bütünleştirilmesi

Tez kapsamında geliştirilen çoklu depo sisteminde kullanılacak sorgu motoru için öncelikle MonARCh'ın kaynak kodu yeni bir depo (repository) altına taşınmış ve kaynak kod, sorgu takibi ile birleştirilmiş SPARQL sorgulama özelliklerinden arındırılarak sadeleştirilmiştir. İşletici birimleri sorgulanacak depolara göre özelleştirilmiş ve kendi kendini zamanlamasına gerek kalmamıştır. Birleştirici birimi, birleştirilmiş Sparql sorgusunu ayrıştırıp İşletici biriminden ziyade Dağıtıcı birimlerine iletiyor iken, sorgu ayrıştırma ortadan kalkmış ve bunun yerine çoklu depo sorgu nesnesindeki birleşik sorgunun içerdiği depolara göre özelleşmiş İşletici birimlerine ilgili sorguları yönlendirmektedir. Böylece bir alt sorgunun SPARQL'da olduğu gibi birden fazla

depo üzerinde çalışması gibi bir gereksinim olmadığından dolayı Dağıtıcı birimi tamamen ortadan kaldırılmıştır. Dolayısıyla, anahtar-değer sorguların işletimi için Scala-Redis istemci bağlayıcısı (client connector)[50], ilişkisel sorguların işletimi için Java-Postgresql istemci bağlayıcısı [51], zaman-serisi sorguların işletimi için InfluxDb-Scala istemci bağlayıcısı [52], arama-belge odaklı sorgular için ise Elasticsearch-Scala istemci bağlayıcısı [53] kütüphaneleri kaynak koda eklenmiştir. Redis, Postgresql, Influxdb ve Elasticsearch için işletici birimleri eklenerek bu veri depoları için sorgu işletimi mümkün hale gelmiştir. Daha sonra her bir İşletici birimi için yine ayrı ayrı Sonuç Dönüştürücü birimleri yazılarak bu veri depolarından gelen sorgu sonuçlarının ortak veri temsil modeli olarak belirlenen RDF'e dönüşümü sağlanmıştır. Bu verilerin birleşim (join) işlemi; MonARCh'ta da kullanılan Paralel Birleşim Yöneticisi ve Karma Birleşim Uygulayıcısı birimleri ile gerçekleştirilip sonucu Birleştirici birimine iletilmektedir. Böylece çoklu depo sorgu işleme döngüsü basit hâliyle tamamlanmaktadır. Geliştirilen sorgu motoru AQuA ile benzerlikler göstermektedir ve MonARCh'ın çekirdeğinden ayrıştırıldığı için hem Scala dilinde yazılmış olup, hem de küme parçalama özelliklerine ve eniyileştirmelerine sahiptir. Dolayısıyla buradan esinlenilerek çoklu depo sorgu motoruna AQuAPooL[54] (Actor based Querying Architecture for Polystores on the Linked data - Çoklu depolar için Bağlı veri üzerinde Aktör tabanlı Sorgulama Mimarisi) adı verilmiştir ve işleyişi Şekil-4.5'te görülmektedir.

Buna göre öncelikle AQuAPooL'un Birleştirici (Federator) aktör birimine çoklu depo ortamındaki (Polystore Environment) depolardan en az birinin sorgulandığı bir "Çoklu Depo Sorgusu" mesajı iletilir (Adım 1). Bu mesaj içerisinde "Sorgu" - "Depo" eşlemelerini tutan bir harita (map) yer almaktadır. Birleştirici bu haritadan faydalanarak her sorguyu kendisiyle bağlantılı olduğu depoda çalıştırmak üzere ilgili İşletici (Executor) aktöre iletir (Adım 2). Sis-

[50]https://github.com/debasishg/scala-redis
[51]https://github.com/pgjdbc/pgjdbc
[52]https://github.com/influxdata/influxdb-client-java/tree/master/client-scala
[53]https://github.com/sksamuel/elastic4s
[54]https://github.com/burakyonyul/aquapool



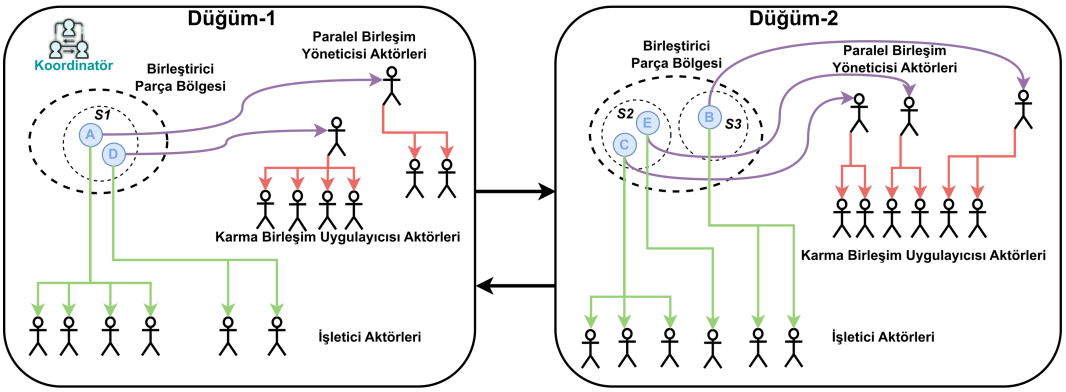
Şekil 4.5: AQuAPooL Mimarisi ve Sorgu İşleme Akışı

temde Redis, Postgresql, InfluxDB ve Elasticsearch olmak üzere 4 adet İşletici aktör tipi bulunmaktadır. İşletici birimi ilgili depoya bağlanarak kendisine gelen sorguyu işletir (Adım 3) ve sonucu elde eder (Adım 4). Depolara kurulan tüm bağlantılar her deponun kendi bağlantı biriminde, "bağlantı havuzu" (connection pooling) yöntemiyle gerçekleştirilir. Böylece sistem yoğun sorgu yükü altındayken bağlantı nesneleri hem tekrar tekrar yaratılmayıp yeniden kullanılırak zaman tasarrufu yapılır, hem de aynı anda kullanılarak paralel sorgu işlemeye olanak tanır. Bir İşletici birimi sorgu sonucunu elde ettikten sonra bu sonucu ilgili depo ile bağlantılı Dönüştürücü birimine iletir (5). İlgili tipteki (Redis, Postgresql, InfluxDB, Elasticsearch) Dönüştürücü birimi sorgu sonucunu ortak veri modeli olarak belirlenen RDF sonucu (RDF Result) biçimine çevirir (Adım 6). İşletici aktör tipine göre (Redis, Postgresql, InfluxDB, Elasticsearch) RDF biçimine dönüştürülmüş sonuç, mesaj olarak Birleştirici aktöre iletilir (Adım 7). Birleştirici birimi elde ettiği dönüştürülmüş sonuçlar içerisinden ortak değişken içerenleri belirleyerek eşleştirir. Eşleşen her sonuç çifti için bir mesaj oluşturularak Paralel Birleşim Yöneticisi aktörüne iletilir (Adım 8). Bu birimde Şekil-4.1'de anlatılan paralel GRACE Karma

Birleşim Algoritması işletilir. Özetlenecek olursa her iki sonuç da kova adı verilen alt kümelere bölünerek birbiriyle eşleşen alt kümeler (Adım 9) paralel olarak Karma Birleşim Uygulayıcı birimlerince karma birleşim işlemine sokulur. Alt kümelerin birleşim sonuçları (Adım 10) toplanarak ana birleşim sonucu elde edilir ve mesaj yoluyla Birleştirici aktöre iletilir (Adım 11). Birleştirici aktör, tüm birleşim işlemleri tamamlandığında çoklu depo sorgusunun esas sonucu oluşturulmuş olur ve mesaj yoluyla istemciye (client) iletilir (Adım 12).

AQuAPooL'un, Akka küme parçalama yöntemiyle oluşturulan örnek küme yapısı modeli, Şekil-4.4'teki MonARCh küme yapısı modeli ile benzerlik göstermektedir. Burada tasvir edilen mimaride sorgu takibi özelliği bulunduğundan, Birleştirici, Dağıtıcı ve İşletici birimlerinin farklı sorgularca yeniden kullanımı mümkün olabilmektedir. Dolayısıyla bu birimler arasından zaten yaratılmış olan bir yapının tekrardan yaratılmasını engellemek adına kendilerine özel Parça Bölgeleri (Shard Regions) tanımlanmış ve böylece sistemdeki kaynakların etkin kullanılması sağlanmıştır. AQuAPooL'da oluşturulan küme yapısında ise yalnızca Birleştirici biriminin Parça Bölgesi bulunmakta olup modeli Şekil-4.6'da görülmektedir. Bunun anlamı da sorgular sisteme yollanan çoklu depo sorgusuna göre bölümlendirilerek düğümler üzerinde dağıtılmaktadır. Daha sonrasında ise belirli bir çoklu depo sorgusu için Birleştirici'nin altında yaratılacak diğer tüm aktörler aynı bilgisayar düğümü üzerinde çalışmaktadır. Bu durum MonARCh'ta farklı bölgelerde yer alan ve birden çok birimce kullanılan parçaların AQuAPooL kapsamında merkezi olarak aynı bilgisayar düğümü üzerinde yönetilmesini sağlayarak ağ maliyetini önemli ölçüde azaltır.

Geliştirilen mimariyi kavramsal olarak değerlendirdiğimizde sorgu motoru ve çoklu depo gerçekleştirimi olarak iki kısımda ele almak mantıklı olacaktır. Buradan hareketle mimarinin isminden de anlaşılacağı üzere AQuA ve PooL kavramları ön plana çıkmaktadır. Her bir sorgu çalıştırıcı ve dönüştürücü farklı AQuA'lar ile temsil edilmektedir. Bunlar kendi başına sorgu işleticisi ve dönüştürücüsünü barındırmaktadır. Dolayısıyla AQuA'lar çoklu depo havu-



Şekil 4.6: AQuAPooL için Örnek Küme Yapısı Modeli

zunda yer alacak her bir depo için bir akışı temsil etmektedir. PooL kelimesi de tüm bu AQuA'ları barındıran ve bağlı veri yardımıyla bütünleştiren çoklu depo havuzu olarak ifade edilebilecektir.

5 DEĞERLENDİRME

Tez çalışmasınca geliştirilen MIMIC-III veri kümesini kullanan çoklu depo sistemi, veri ortamı ve sorgu işleme bakış açılarından Bölüm-4.1.2'deki veri kümesi çözümlemesi teknikleri, Bölüm-4.2.1'deki son teknoloji (state of the art) depoların seçimi ve Bölüm-3.1'de ortaya konan literatürden ölçeklenebilir ve esnek çoklu depolar yaratabilmek için en iyi yöntemler (best practices) kullanılarak, kapsamlı çözümlemeler ve yüksek lisans öğreniminden bu yana edinilen teorik ve teknik deneyimler ile bu kapsamda yapılan çalışmaların sonucunda ortaya çıkmıştır. Dolayısıyla çoklu depo ortamı ve sorgu motoru olmak üzere iki ana bileşen ve bunları geliştirmek için kullanılan çok sayıda alt bileşenden oluştuğundan değerlendirme noktasında birçok bakış açısını hesaba katmak gerekir.

Öncelikle bölünecek verinin seçilen çoklu depo ortamına doğru şekilde bölümlendiğini sınamak gerekmektedir. Bunun yanında ise seçilen saklama çözümlerinin beklenen performansı gösterip göstermediğini dolayısıyla da istenen çözümü sunup sunmadığını ortaya çıkarmak gerekir. Böylece hem verinin doğru bölümlendiği hem de bölümlenen bu veri için uygun deponun seçildiği ortaya konmuş olur. Bu kapsamda sistemin gelen sorguları performanslı bir şekilde çalıştırdığı gözler önüne serilmelidir. Ayrıca sorgu işletim performansının yalnızca tek depo üzerinden sunulan bir çözüme kıyasla daha yüksek olduğunun ortaya konması çoklu depo sistemi yaklaşımının gerekliliğini ispatlayacaktır. Bunun dışında sunulan çözüm hem yoğun sorgu yükü altında etkin ve performanslı çalışabilmeli, hem de bu sorguları ölçeklenebilir şekilde işleyebilmelidir.

Literatürde çoklu depo değerlendirmesi ile ilgili en güncel çalışma (Karimov et al., 2019) tarafından bir kıyaslama noktası (benchmark) tasarımı altında PolyBench yayınında ortaya konmuştur. Bankacılık üzerine kurulan sentetik bir veri kümesi üzerinde üç temel kullanım durumu (sorgu) ve iki test senaryosu belirlenmiştir. Benzer olarak (Poudel et al., 2022) yaptıkları çalışmada büyük ölçekli astronomi verisi üzerinde yaratmış oldukları bir-

birinden farklı seçicilikte ve özelliklerdeki 20 sorgu ile bir değerlendirme yapmışlardır. Tezde de ana fikir olarak eldeki veri ve kullanım durumu üzerine oluşturulan bir çoklu depo sistemi ve değerlendirme tasarımı üzerinde durulmuştur. Fakat tezin literatürdeki benzer çalışmalardan farkını ortaya çıkaran en önemli nokta, kritik öneme sahip MIMIC-III veri kümesi üzerine kurulmuş olması ve hem sistemin hem de değerlendirmenin bu veri kümesini kullanan gerçekçi medikal senaryolara dayanıyor olmasıdır. Bu medikal senaryolardan zengin ve çok çeşitli bir sorgu kümesi oluşturulmuştur. Böylece yukarıda değerlendirme bakış açısından bahsi geçen bu iki çalışmanın da güçlü yanlarından faydalanıldığı söylenebilecektir.

Bölüm-5.1 öncelikle güncel çoklu depo sistemleri ile AQuAPooL özellikleri bakımından kıyaslanacaktır. İlerleyen bölümde (bkz. Bölüm-5.2) ilk olarak sistemin değerlendirilmesi için, çoklu depo sisteminin inşa edildiği MIMIC-III veri kümesi üzerinde bir sınama sorgu kümesinin oluşturulması anlatılacaktır. Devamında Bölüm-5.3'te işletilen sorgu sonuçları tablolar ve grafikler cinsinden verilerek çözümlenecek ve yorumlanacaktır. Bölüm-5.4'te sistemin ölçeklenebilirlik parametreleri ve bunu nasıl sağladığı tablo ve grafikler ile yorumlanacak olup, Bölüm-5.5'te ise sistemin farklı bakış açılarından kısıtları ortaya konarak tartışılacaktır.

5.1 Çoklu Depo Sistemlerinin Özellik Bakımından Kıyaslanması

Bölüm-3'te değinilen güncel ve önemli çoklu depo sistemleri ile tez kapsamında geliştirilen AQuAPooL aracının sahip oldukları önemli özellikler bakımından kıyaslanması Tablo-5.1'de görülmektedir. Burada sistemler son teknoloji bir sistemde olması gereken birçok farklı bakış açısından değerlendirilmiştir. Öncelikle sorgulama kategorisinde sistemlerin sorguları işletebilmek için ortak bir sorgu arayüzüne sahip olmaları ve sorgu işletimini depoların yerel sorguları üzerinden yapabilmeleri değerlendirilmiştir. Ortak sorgu arayüzünün bulunması tek bir noktadan sorgulama kolaylığı sağlarken,

yerel sorguların işletiminin desteklenmesi ise depoya göre eniyileştirilmiş sorguların yazılmasına olanak tanımaktadır. AQuAPooL şu an için yalnızca yerel sorguların işletimini desteklemekte olup, gelecek ilk çalışma olarak SQL tabanlı ortak bir sorgu arayüzünün geliştirilmesi planlanmaktadır.

Desteklenen veri deposu bakış açısından tüm sistemlerin hem ilişkisel hem de NoSQL veri depoları ile çalışabildikleri görülmektedir. Bir diğer önemli kategoride ise sistemlerin eşzamanlı, paralel ve dağıtık sorgu işleme ve birleşim yapabilme yetenekleri irdelenmiştir. Bu kavramlar sırasıyla açıklanacak olursa:

- *Eşzamanlı*: Farklı görevleri (task) aynı anda yerine getirebilme yeteneği olarak ifade edilebilir.
- *Paralel*: Bir görevin birden çok alt göreve bölünerek, bu alt görevlerin aynı anda işletilmesidir.
- *Dağıtık*: Görevlerin, birbirleriyle ileti geçişi (message passing) yoluyla iletişim kuran birden fazla makinenin kaynaklarını kullanarak yerine getirilmesidir.

Tüm bu özellikler bir çoklu depo sisteminin verimini (throughput) artırarak ve sorgu işletim süresini kısaltarak performansını yükseltmektedir. Tablodan görüldüğü üzere bu yeteneklerin tamamından ziyade en azından birine dahi sahip sistemlerin sayısı kısıtlıdır. Dahası, AQuAPooL'un aktör model ile farklı görevleri eşzamanlı çalıştırabilmesi, sorgu işleme ve birleşim görevlerini alt parçalara bölerek çoklu işlemci çekirdekleri üzerinde paralel olarak işletebilmesi ve tüm bu işlemleri küme üzerinde dağıtık olarak yerine getirebilmesi sayesinde üç özelliği de sağlayan tek çoklu depo sistemi olduğu açıkça görülmektedir ve bu da sistemin en önemli özgün yanlarından birini oluşturmaktadır.

Sistemlerin kıyaslandığı özgün ve farklı bir diğer özellik ise anlamsal veri bütünleştirme yeteneğidir. Bağlı veri ilkelerinin 5 yıldızlı ve yapısal açıdan zengin veriyi tanımlaması ile RDF şemanın bütünleştirici özelliği sayesinde çoklu depo sistemlerinde RDF ortak veri modeli üzerinden anlamsal bir veri bütünleştirme sağlanabilmektedir. Böylece çoklu depo sistemleri bağlı verinin tüm avantajlarından faydalanabilir ve açık bağlı veri bulutu ile

iletişim kurabilir duruma gelmektedir. Tablodan görüleceği gibi anlamsal veri bütünleştirme yeteneği kıyaslanan tüm bu sistemlerin yalnızca üç tanesinde bulunmakta olup, AQuAPooL'un diğer bir özgün yanıdır.

Son olarak olgunlaşmamış birkaç çalışma dışındaki tüm sistemler oluşturulan veya hazır olarak seçilen test ortamlarında farklı bakış açılarından sistematik olarak değerlendirilmiştir. Ayrıca yalnızca sorgu diline, anlamsallığa ve kavramsallığa odaklanan çalışmalar dışındaki tüm sistemler ölçeklenebilir olarak tasarlanmmış olup gerektiğinde kaynak eklenerek sorgu işleme kapasitesini ve performansını artırabilmektedir. AQuAPooL diğer çoklu depo çalışmalarına kıyasla en fazla değerlendirme ölçütünü karşılayan sistem olarak öne çıkmakta olup, ortak sorgu arayüzü dışındaki tüm özelliklere sahiptir ve yakın gelecekte de bu yeteneği de kapsayacak hâle gelecektir.

	Sorgulama		Desteklenen Veri Deposu		Sorgu İşleme/Birleşim Yeteneği					
Sistem	Ortak Sorgu Arayüzü	Yerel Sorgular	İlişkisel	NoSQL	Eşzamanlı	Paralel	Dağıtık	Anlamsal Veri Bütünleştirme	Sistematik Değerlendirme	Ölçeklenebilirlik
AQuAPooL		X	X	X	X	X	X	X	X	X
CloudMDSQL	X		X	X		X	X		X	X
BigDAWG	X	X	X	X					X	
Myria	X		X	X		X	X		X	X
Apache Drill	X		X	X		X	X		X	X
ESTOCADA	X	X	X	X					X	X
FORWARD	X		X	X						
AWESOME	X		X	X		X			X	X
HYBRID.POLY	X		X	X				X		
TATOOINE		X	X	X						
RHEEM (Apache Wayang)	X	X	X	X		X	X		X	X
PolyWeb	X		X	X				X	X	
Polypheny-DB	X	X	X	X						X
Poudel et. al, (2019, 2022)	X	X	X	X					X	X

Tablo 5.1: AQuAPooL ve Güncel Çoklu Depo Sistemlerinin Son Teknoloji Özellikler ve Gereksinimler Bakımından Kıyaslanması

5.2 Sınama Sorgu Kümesinin Oluşturulması

Bölüm-4.1.5'te MIMIC-III (Johnson et al., 2016) yayınına referans veren (genellikle yapay zekâ ile ilişkili) çalışmaların çözümlenmesi sonucu çıkarılan senaryolar ve bu senaryolarda kullanılan MIMIC-III veri kümesinin alt parçaları detaylandırılmıştı. Buna göre çıkarılan bu senaryoların ait olduğu çalışmalardan veri kullanımı açısından önemli olanları tespit edilerek, bu

çalışmaların ilgili tespite ulaşmak için hangi MIMIC-III verilerini ve bunların alt parçalarını kullandıkları belirlenmiştir. Daha sonra bu veriler ve alt parçaları çözümlenerek sorgu bileşenleri bulunur. Bulunan sorgu bileşenleri senaryolar ile uyumlu olarak biraraya getirilerek tekil ve birleşik sorgular oluşturulmuştur. Böylece elde edilecek bu sorgu sonuçlarıyla tezde ortaya konan çoklu depo sistemin önemi vurgulanacak olup aynı zamanda kullanılan MIMIC-III verisinin de çoklu depo bölümlendirmesine uygun olup olmadığı gözler önüne serilmiş olacaktır.

5.2.1 Önemli Kullanım Senaryolarının Seçimi

Bölüm-4.1.5'te anlatıldığı üzere MIMIC-III veri kümesi sağlık alanında önemli bir yere sahip olup, hem tıp, hem bilgisayar bilimleri, hem de bu iki disiplini bir arada kullanan çalışmalar için bir temel teşkil etmektedir. Çok farklı bakış açılarından 6000'e yakın çalışma veri kümesine referans vermiş olup bunların sayıları her geçen gün katlanarak artmaktadır. Bu tespit yapılırken en çok çalışılan senaryolar ve bu senaryoların çalışıldığı yayınlar içerisinden en fazla referans alınanlar seçilmiş ve bu da sorgu kümesi oluşturulurken bir temel teşkil etmiştir.

Sonuç olarak tüm bu senaryolar arasından sorgu kümesi oluşturulması için sepsis (kan zehirlenmesi) tespiti, ölüm oranı tahminleme (mortality prediction) ve klinik metin sınıflandırması (clinical text classification) olmak üzere üç temel kullanım durumu belirlenmiştir. Bahsi geçen senaryoların incelemesi aşağıda verilmektedir:

- **SEPSIS TESPİT EDİLMESİ:** Kan zehirlenmesi teşhisi (Raghu et al., 2017; Komorowski et al., 2018; Nemati et al., 2018; Wu et al., 2018; Desautels et al., 2016; Kong et al., 2020) hastanede yoğun bakım birimlerinde en sık rastlanan ölüm nedenlerinden birisi olması nedeniyle önem arz etmektedir. (Chartevents, Inputevents, Outputevents, Labevents)
  - EMR (electronic medical record) Verisi: Arterial blood pressure, Systolic pressure, Respiratory rate, Heart Rate, SpO2, Temperature,

GCS

  - Demografik/Statik Veriler: Readmission, GCS, Gender, Sofa Score
  - Laboratuvar Değerleri: Albumin, Arterial pH, Calcium, Blood Glucose, Hemoglobin, Magnesium, PTT, Potassium, SGPT, ABG, BUN, Chloride, Bicarbonate, INR, Sodium, CO2, Creatinine, Ionized Calcium, Prothrombin, Platelets, SGOT, Total Bilirubin
  - Yaşamsal Belirtiler: Blood pressure, PaCO2, PaO2, FiO2, Respiratory Rate, Temperature, Weight, Heart Rate, SpO2
  - Alım ve Çıktı Olayları: Fluid Output, Mechanical Ventilation

- **ÖLÜM ORANI TAHMİNLEME:** Hastane içi ölüm oranının (mortality prediction) bulunması (Johnson et al., 2017; Rajkomar et al., 2018; Che et al., 2018; Harutyunyan et al., 2019) tedavilerin ve müdahalelerin başarısı açısından önemlidir. (Chartevents, Inputevents, Outputevents, Labevents, Noteevents)
  - Heart rate, Carbon Dioxide, Blood pressure, Respiratory rate, Temperature, Peripheral Oxygen Saturation, Glucose, Glasgow coma scale, Glasgow coma scale components (motor, verbal, eyes), pH, Base excess, Bicarbonate, Hematocrit, Hemoglobin, Chloride, Calcium, Temperature, Potassium, Sodium, Lactate, Anion gap, Albumin, Bicarbonate, Bilirubin, Creatinine, Platelet, Prothrombin time, Urine output, Capillary Refill Rate, Fraction Inspired Oxygen, Height, Weight
  - Input events (hastaya verilen sıvılar: örn. insulin), output events (hastadan alınan sıvılar: örn. urine), lab events (laboratuvar test sonuçları: örn. pH, Platelet count) ve prescription events (doktorlar tarafından reçetelendirilen ilaçlar: örn. aspirin and potassium chloride), Cholesterol, TroponinI, TroponinT, ALP, ALT, AST, SaO2, WBC, HCO3, HCT, Urine, MAP, DiasABP, SysABP
  - NEWS'in fizyolojik parametreleri artı ek olanlar: herhangi bir ek oksijen desteği, AVPU sistemini kullanarak bilinç seviyesi, beyaz

kan hücresi sayısı

- **KLİNİK METİN SINIFLANDIRMASI:** Hasta notlarının analiz edilerek hastalıkların, tanıların, tedavilerin vb. sınıflandırılması amaçlanmaktadır (Mullenbach et al., 2018; Prakash et al., 2017; Luo, 2017). (Noteevents)
  - İlişki Sınıflandırma (Luo, 2017): Metin Kutusu-1'de örneklendirilmiştir.
  - Klinik metinlerden (taburcu özetlerinden) ICD-9 etiketlemesi (Mullenbach et al., 2018): Metin Kutusu-2'de örneklendirilmiştir.
  - Serbest biçimli metinden klinik durum teşhisi (Prakash et al., 2017): Metin Kutusu-3'de örneklendirilmiştir.

Tıbbi Problemler ve Tedaviler Arası İlişkiler TrP:

- Tıbbi problem için uygulanan tedavi (TrAP). Örneğin, "he was given Entresto to treat his high blood pressure".
- Tıbbi sorun nedeniyle tedavi uygulanmıyor (TrNAP). Örneğin, "Relafen which is contraindicated because of ulcers".
- Tedavi, tıbbi sorunu (TrIP) iyileştirir. (TrIP). Örneğin, "infection resolved with a full course of cephalexin"
- Tedavi tıbbi soruna (TrCP) neden olur. (TrCP). Örneğin, "the patient took amoxicillin for two days, which caused diarrhea"
- Uygulanmakta olan bir tedavi (TrWP) nedeniyle veya uygulanmasına rağmen hastanın tıbbi sorununun kötüleşmesi. (TrWP). Örneğin, "the tumor was growing despite the drain"
- Tedavi, maddede belirtildiği gibi tıbbi sorunla ilgili değildir. (None)

Tıbbi sorunlar ve TeP testleri arasındaki ilişkiler TeP:

- Test bazı tıbbi sorunları ortaya çıkardı (TeRP). Örneğin, "an echocardiogram revealed a pericardial effusion"
- Tıbbi bir sorunu (TeCP) araştırmak için test yapıldı. (TeCP). Örneğin, "chest X-ray done to rule out pneumonia"
- Testin metinde belirtildiği gibi tıbbi sorunla ilgisi yok (None)

Tıbbi problemler ve tıbbi problemler arasındaki ilişkiler: (PP):

- İki problem birbiriyle ilişkilidir (PIP). Örneğin, "Azotemia presumed secondary to sepsis"
- Tıbbi sorun metinde belirtildiği gibi tıbbi sorunla ilgili değil (None)

MetinKutusu 1: İlişki Sınıflandırma Örneği

934.1: “Foreign body in main bronchus” CAML (HI)
Cosine Sim
CNN
Logistic Regression
...line placed bronchoscopy performed showing **large mucus plug on** the left on transfer to...
...also needed medication to help **your body maintain your** blood pressure after receiving iv...
...found to have a large **lll lingular pneumonia on** chest x ray he was...
...impression confluent consolidation involving nearly **the entire left** lung with either broncho- centric or vascular...

442.84: “Aneurysm of other visceral artery”
CAML (I)
Cosine Sim CNN
...and gelfoam embolization of right **hepatic artery branch pseudoaneurysm** coil embolization of the gastroduodenal...
...coil embolization of the gastroduodenal **artery history of present** illness the pt is a...
...foley for hemodynamic monitoring and **serial hematocrits angio was** performed and his gda was...

Logistic Regression (I)
428.20: “Systolic heart failure, unspecified”
CAML Cosine Sim CNN
Logistic Regression (HI)
...no mitral valve prolapse moderate **to severe mitral regurgitation** is seen the tricuspid valve...
...is seen the estimated pulmonary **artery systolic pressure is** normal there is no pericardial...
...and suggested starting hydralazine imdur **continue aspirin arg admitted** at baseline cr appears patient...
...anticoagulation monitored on tele pump **systolic dysfunction with ef** of seen on recent echo...

MetinKutusu 2: Klinik metinlerden örnek ICD9 Etiketlemesi Örneği

MIMIC-III Note
Medical Note (partially shown)
Cardiac arrest
DateofBirth:[**2606–2–28**] Sex:M
Service: Medicine
Chief Complaint:
Admitted from rehabilitation for hypotension (systolic blood pressure to the 70s) and decreased urine output. History of present illness:

The patient is a 76-year-old male who had been hospitalized at the [**Hospital1 3007**] from [**8–29**] through [**9–6**] of 2002 after undergoing a left femoral-AT bypass graft and was subsequently discharged to a rehabilitation facility.

On [**2682–9–7**], he presented again to the [**Hospital1 3087**] after being found to have a systolic blood pressure in the 70s and no urine output for 17 hours.

Diagnosis
Cardiorespiratory arrest. Non-Q-wave myocardial infarction. Acute renal failure.

Relevant Wikipedia Page
Cardiac arrest

Cardiac arrest is a sudden stop in effective blood circulation due to the failure of the heart to contract effectively or at all[1]. A cardiac arrest is different from (but may be caused by) a my- ocardial infarction (also known as a heart attack), where blood flow to the muscle of the heart is impaired such that part or all of the heart tissue dies...

Signs and symptoms

Cardiac arrest is sometimes preceded by certain symptoms such as fainting, fatigue, blackouts, dizziness, chest pain, shortness of breath, weakness, and vomiting. The arrest may also occur with no warning...

MetinKutusu 3: Serbest Biçimdeki Metinlerden Klinik Durum Teşhisi Örneği

5.2.2 Kullanım Senaryolarından Sorgu Çıkarımı

Bölüm-5.2.1'de medikal senaryolar sepsis tespiti, ölüm oranı tahminleme ve klinik metin sınıflandırması olarak seçilmişti. Chartevents ve Noteevents bu kullanım durumlarında en çok kullanılan tablolar olmakla birlikte Inputevents, Outputevents ve Labevents tablolarından da faydalanılmaktadır. Buna göre Hastaların yatış esnasında izlenen yaşamsal değerleri sırasıyla her bir senaryoda kullanılan metriklerin tanımlayıcıları (ID) D_Items tablosundan, hastaların tanıları ve tedavileri ile ilgili ICD_9 kodları da D_ICD_DIAGNOSES ve D_ICD_PROCEDURES tablolarından SQL LIKE sorgusu ile bulunarak sorguların oluşturulabilmesi için yapıtaşları elde edilmiştir. Daha sonra Chartevents, Procedures_ICD, Diagnoses_ICD ve Noteevents tablolarından belirlenen metriklerle ilgili yapıtaşları kullanılarak, senaryolar ile bağlantılı 28 adet basit ve 7 adet birleşik olmak üzere toplam 35 adet sorgu oluşturulmuştur.

5.2.2.1 Zaman Serisi Tekil Sorgular

Bu bölümde çoklu depo ortamında zaman serisi olarak InfluxDB'de saklanan veriler için oluşturulan sorgular daha sonra kıyaslanmak üzere SQL eşlenikleri ile birlikte listelenmektedir.

Verilen tarih aralığındaki arter kan basıncı (Arterial Blood Pressure systolic) değerini ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-8'de ve zaman serisi veritabanı InfluxDB'den elde eden Flux sorgusu ise Sorgu-9'da görülmektedir.

```
SELECT * FROM mimiciii.chartevents WHERE itemid=220050
AND (charttime BETWEEN '2200-01-01 00:00:00' AND '2209-12-31
    23:59:00')
```

Sorgu 8: "Arterial Blood Pressure systolic" ölçütünün 01.01.2011 - 31.12.2012 aralığındaki değerlerini elde eden SQL sorgusu

```
from(bucket: "mimic-iii")
 |> range(start:2011-01-01, stop:2012-12-31)
 |> filter(fn: (r) => r["_measurement"] == "chart_event")
```

```
|> filter(fn: (r) => r["itemid"] == "220050")
|> filter(fn: (r) => r["_field"] == "value" or r["_field"]
  == "icustay_id" or r["_field"] == "hadm_id" or r["_tag"]
  == "subject_id")
|> pivot(rowKey: ["_time"], columnKey: ["_field"],
  valueColumn: "_value")
```

Sorgu 9: "Arterial Blood Pressure systolic" ölçütünün 01.01.2011 - 31.12.2012 aralığındaki değerlerini elde eden Flux sorgusu

Bir sepsis hastasının nabız (Heart Rate) değerini ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-10'da ve zaman serisi veritabanı InfluxDB'den elde eden Flux sorgusu ise Sorgu-11'de görülmektedir.

```
SELECT * FROM mimiciii.chartevents WHERE subject_id=191 AND
  itemid=211
```

Sorgu 10: Bir Sepsis Hastasının "Heart Rate" ölçütünün değerlerini elde eden SQL sorgusu

```
from(bucket: "mimic-iii")
|> range(start:2000-01-01, stop:2012-12-31)
|> filter(fn: (r) => r["_measurement"] == "chart_event")
|> filter(fn: (r) => r["itemid"] == "211")
|> filter(fn: (r) => r["subject_id"] == "191")
|> filter(fn: (r) => r["_field"] == "value" or r["_field"]
  == "icustay_id" or r["_field"] == "hadm_id")
|> pivot(rowKey: ["_time"], columnKey: ["_field"],
  valueColumn: "_value")
```

Sorgu 11: Bir Sepsis Hastasının "Heart Rate" ölçütünün değerlerini elde eden Flux sorgusu

Bir sepsis hastasının solunum hızı (Respiratory Rate) değerini ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-12'de ve zaman serisi veritabanı InfluxDB'den elde eden Flux sorgusu ise Sorgu-13'te görülmektedir.

```
SELECT * FROM mimiciii.chartevents WHERE subject_id=191 AND
    itemid=618
```

Sorgu 12: Bir Sepsis Hastasının "Respiratory Rate" ölçütünün değerlerini elde eden SQL sorgusu

```
from(bucket: "mimic-iii")
 |> range(start:2000-01-01, stop:2012-12-31)
 |> filter(fn: (r) => r["_measurement"] == "chart_event")
 |> filter(fn: (r) => r["itemid"] == "618")
 |> filter(fn: (r) => r["subject_id"] == "191")
 |> filter(fn: (r) => r["_field"] == "value" or r["_field"]
   == "icustay_id" or r["_field"] == "hadm_id")
 |> pivot(rowKey: ["_time"], columnKey: ["_field"],
   valueColumn: "_value")
```

Sorgu 13: Bir Sepsis Hastasının "Respiratory Rate" ölçütünün değerlerini elde eden Flux sorgusu

Bir sepsis hastasının oksijen satürasyonu (SpO2) değerini ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-14'te ve zaman serisi veritabanı InfluxDB'den elde eden Flux sorgusu ise Sorgu-15'te görülmektedir.

```
SELECT * FROM mimiciii.chartevents WHERE subject_id=191 AND
    itemid=646
```

Sorgu 14: Bir Sepsis Hastasının "SpO2" ölçütünün değerlerini elde eden SQL sorgusu

```
from(bucket: "mimic-iii")
 |> range(start:2000-01-01, stop:2012-12-31)
 |> filter(fn: (r) => r["_measurement"] == "chart_event")
 |> filter(fn: (r) => r["itemid"] == "646")
 |> filter(fn: (r) => r["subject_id"] == "191")
 |> filter(fn: (r) => r["_field"] == "value" or r["_field"]
   == "icustay_id" or r["_field"] == "hadm_id")
```

```
|> pivot(rowKey: ["_time"], columnKey: ["_field"],
  valueColumn: "_value")
```

Sorgu 15: Bir Sepsis Hastasının "SpO2" ölçütünün değerlerini elde eden Flux sorgusu

Bir sepsis hastasının Fahrenheit cinsinden vücut sıcaklığı (Temperature F) değerini ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-16'da ve zaman serisi veritabanı InfluxDB'den elde eden Flux sorgusu ise Sorgu-17'de görülmektedir.

```
SELECT * FROM mimiciii.chartevents WHERE subject_id=191 AND
  itemid=678
```

Sorgu 16: Bir Sepsis Hastasının "Temperature F" ölçütünün değerlerini elde eden SQL sorgusu

```
from(bucket: "mimic-iii")
 |> range(start:2000-01-01, stop:2012-12-31)
 |> filter(fn: (r) => r["_measurement"] == "chart_event")
 |> filter(fn: (r) => r["itemid"] == "678")
 |> filter(fn: (r) => r["subject_id"] == "191")
 |> filter(fn: (r) => r["_field"] == "value" or r["_field"]
   == "icustay_id" or r["_field"] == "hadm_id")
 |> pivot(rowKey: ["_time"], columnKey: ["_field"],
   valueColumn: "_value")
```

Sorgu 17: Bir Sepsis Hastasının "Temperature F" ölçütünün değerlerini elde eden Flux sorgusu

Bir sepsis hastasının Glasgow Koma Ölçeği Toplamı (Glasgow Coma Scale - GCS Total) değerini ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-18'de ve zaman serisi veritabanı InfluxDB'den elde eden Flux sorgusu ise Sorgu-19'da görülmektedir.

```
SELECT * FROM mimiciii.chartevents WHERE subject_id=191 AND
```

```
    itemid=198
```

Sorgu 18: Bir Sepsis Hastasının "GCS Total" ölçütünün değerlerini elde eden SQL sorgusu

```
from(bucket: "mimic-iii")
 |> range(start:2000-01-01, stop:2012-12-31)
 |> filter(fn: (r) => r["_measurement"] == "chart_event")
 |> filter(fn: (r) => r["itemid"] == "198")
 |> filter(fn: (r) => r["subject_id"] == "191")
 |> filter(fn: (r) => r["_field"] == "value" or r["_field"]
   == "icustay_id" or r["_field"] == "hadm_id")
 |> pivot(rowKey: ["_time"], columnKey: ["_field"],
   valueColumn: "_value")
```

Sorgu 19: Bir Sepsis Hastasının "GCS Total" ölçütünün değerlerini elde eden Flux sorgusu

Bir sepsis hastasının nabız (Heart Rate) ve solunum hızı (Respiratory Rate) değerini ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-20'de ve zaman serisi veritabanı InfluxDB'den elde eden Flux sorgusu ise Sorgu-21'de görülmektedir.

```
SELECT * FROM mimiciii.chartevents WHERE chartevents.
    subject_id=191 AND (chartevents.itemid=211 OR chartevents
    .itemid=618)
```

Sorgu 20: Bir Sepsis Hastasının "Heart Rate" ve "Respiratory Rate" ölçütünün değerlerini elde eden SQL sorgusu

```
from(bucket: "mimic-iii")
|> range(start:2000-01-01, stop:2012-12-31)
|> filter(fn: (r) => r["_measurement"] == "chart_event")
|> filter(fn: (r) => r["itemid"] == "211" or r["itemid"] ==
   "618")
|> filter(fn: (r) => r["subject_id"] == "191")
```

```
|> filter(fn: (r) => r["_field"] == "value" or r["_field"]
   == "icustay_id" or r["_field"] == "hadm_id")
|> pivot(rowKey: ["_time"], columnKey: ["_field"],
   valueColumn: "_value")
```

Sorgu 21: Bir Sepsis Hastasının "Heart Rate" ve "Respiratory Rate" ölçütünün değerlerini elde eden Flux sorgusu

Bir sepsis hastasının nabız (Heart Rate), solunum hızı (Respiratory Rate) ve oksijen satürasyonu (SpO2) değerini ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-22'de ve zaman serisi veritabanı InfluxDB'den elde eden Flux sorgusu ise Sorgu-23'te görülmektedir.

```
SELECT * FROM mimiciii.chartevents WHERE chartevents.
   subject_id=191 AND (chartevents.itemid=211 OR chartevents
   .itemid=618 OR chartevents.itemid=646)
```

Sorgu 22: Bir Sepsis Hastasının "Heart Rate", "Respiratory Rate" ve "SpO2" ölçütünün değerlerini elde eden SQL sorgusu

```
from(bucket: "mimic-iii")
|> range(start:2000-01-01, stop:2012-12-31)
|> filter(fn: (r) => r["_measurement"] == "chart_event")
|> filter(fn: (r) => r["itemid"] == "211" or r["itemid"] ==
   "618" or r["itemid"] == "646")
|> filter(fn: (r) => r["subject_id"] == "191")
|> filter(fn: (r) => r["_field"] == "value" or r["_field"]
   == "icustay_id" or r["_field"] == "hadm_id")
|> pivot(rowKey: ["_time"], columnKey: ["_field"],
   valueColumn: "_value")
```

Sorgu 23: Bir Sepsis Hastasının "Heart Rate", "Respiratory Rate" ve "SpO2" ölçütünün değerlerini elde eden Flux sorgusu

Bir sepsis hastasının nabız (Heart Rate), solunum hızı (Respiratory Rate), oksijen satürasyonu (SpO2) ve Fahrenheit cinsinden vücut sıcaklığı (Temperature F) değerini ilişkisel veritabanı Postgresql'den elde eden SQL

sorgusu Sorgu-24'te ve zaman serisi veritabanı InfluxDB'den elde eden Flux sorgusu ise Sorgu-25'te görülmektedir.

```
SELECT * FROM mimiciii.chartevents WHERE chartevents.
    subject_id=191 AND (chartevents.itemid=211 OR chartevents
    .itemid=618 OR chartevents.itemid=646 OR chartevents.
    itemid=678)
```

Sorgu 24: Bir Sepsis Hastasının "Heart Rate", "Respiratory Rate", "SpO2" ve "Temperature F" ölçütünün değerlerini elde eden SQL sorgusu

```
from(bucket: "mimic-iii")
|> range(start:2000-01-01, stop:2012-12-31)
|> filter(fn: (r) => r["_measurement"] == "chart_event")
|> filter(fn: (r) => r["itemid"] == "211" or r["itemid"] ==
   "618" or r["itemid"] == "646" or r["itemid"] == "678")
|> filter(fn: (r) => r["subject_id"] == "191")
|> filter(fn: (r) => r["_field"] == "value" or r["_field"]
   == "icustay_id" or r["_field"] == "hadm_id")
|> pivot(rowKey: ["_time"], columnKey: ["_field"],
   valueColumn: "_value")
```

Sorgu 25: Bir Sepsis Hastasının "Heart Rate", "Respiratory Rate", "SpO2" ve "Temperature F" ölçütünün değerlerini elde eden Flux sorgusu

Bir sepsis hastasının nabız (Heart Rate), solunum hızı (Respiratory Rate), oksijen satürasyonu (SpO2), Fahrenheit cinsinden vücut sıcaklığı (Temperature F) ve Glasgow Koma Ölçeği Toplamı (Glasgow Coma Scale - GCS Total) değerini ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-26'da ve zaman serisi veritabanı InfluxDB'den elde eden Flux sorgusu ise Sorgu-27'de görülmektedir.

```
SELECT * FROM mimiciii.chartevents WHERE chartevents.
    subject_id=191 AND (chartevents.itemid=211 OR chartevents
    .itemid=618 OR chartevents.itemid=646 OR chartevents.
```

```
    itemid=678)
```

Sorgu 26: Bir Sepsis Hastasının "Heart Rate", "Respiratory Rate", "SpO2", "Temperature F" ve "GCS Total" ölçütünün değerlerini elde eden SQL sorgusu

```
from(bucket: "mimic-iii")
|> range(start:2000-01-01, stop:2012-12-31)
|> filter(fn: (r) => r["_measurement"] == "chart_event")
|> filter(fn: (r) => r["itemid"] == "211" or r["itemid"] ==
    "618" or r["itemid"] == "646" or r["itemid"] == "678")
|> filter(fn: (r) => r["subject_id"] == "191")
|> filter(fn: (r) => r["_field"] == "value" or r["_field"]
    == "icustay_id" or r["_field"] == "hadm_id")
|> pivot(rowKey: ["_time"], columnKey: ["_field"],
    valueColumn: "_value")
```

Sorgu 27: Bir Sepsis Hastasının "Heart Rate", "Respiratory Rate", "SpO2", "Temperature F" ve "GCS Total" ölçütünün değerlerini elde eden Flux sorgusu

Bir sepsis hastasının tüm yatış çizelgesi değerlerini ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-28'de ve zaman serisi veritabanı InfluxDB'den elde eden Flux sorgusu ise Sorgu-29'da görülmektedir.

```
SELECT * FROM mimiciii.chartevents WHERE chartevents.
    subject_id=191
```

Sorgu 28: Bir Sepsis Hastasının tüm yatış çizelgesi ölçüt değerlerini elde eden SQL sorgusu

```
from(bucket: "mimic-iii")
|> range(start:2000-01-01, stop:2012-12-31)
|> filter(fn: (r) => r["_measurement"] == "chart_event")
|> filter(fn: (r) => r["subject_id"] == "191")
|> filter(fn: (r) => r["_field"] == "value" or r["_field"]
    == "icustay_id" or r["_field"] == "hadm_id" or r["_tag"]
    == "itemid")
```

```
|> pivot(rowKey: ["_time"], columnKey: ["_field"],
   valueColumn: "_value")
```

Sorgu 29: Bir Sepsis Hastasının tüm yatış çizelgesi ölçüt değerlerini elde eden Flux sorgusu

Bir sepsis hastasının belirli bir saat aralığındaki tüm yatış çizelgesi değerlerinin ilişkisel veritabanı Postgresql'den elde eden SQL sorguları Sorgu-30 ve 32'de ve zaman serisi veritabanı InfluxDB'den elde eden Flux sorgusu ise Sorgu-31 ve 33'te görülmektedir.

```
SELECT chartevents.* FROM mimiciii.chartevents WHERE
   chartevents.subject_id=191 AND (charttime BETWEEN '
   2196-04-10 08:00:00' AND '2196-04-10 12:00:00')
```

Sorgu 30: Bir Sepsis Hastasının belirli bir saat aralığındaki tüm yatış çizelgesi ölçüt değerlerini elde eden SQL sorgusu

```
from(bucket: "mimic-iii")
|> range(start:2010-04-10T08:00:00Z, stop:2010-04-10T12
   :00:00Z)
|> filter(fn: (r) => r["_measurement"] == "chart_event")
|> filter(fn: (r) => r["subject_id"] == "191")
|> filter(fn: (r) => r["_field"] == "value" or r["_field"]
   == "icustay_id" or r["_field"] == "hadm_id")
|> pivot(rowKey: ["_time"], columnKey: ["_field"],
   valueColumn: "_value")
```

Sorgu 31: Bir Sepsis Hastasının belirli bir saat aralığındaki tüm yatış çizelgesi ölçüt değerlerini elde eden Flux sorgusu

```
SELECT chartevents.* FROM mimiciii.chartevents WHERE
   chartevents.subject_id=191 AND (charttime BETWEEN '
   2196-04-10 11:00:00' AND '2196-04-10 15:00:00')
```

Sorgu 32: Bir Sepsis Hastasının belirli bir saat aralığındaki tüm yatış çizelgesi ölçüt değerlerini elde eden SQL sorgusu

```
from(bucket: "mimic-iii")
|> range(start:2010-04-10T11:00:00Z, stop:2010-04-10T15
   :00:00Z)
|> filter(fn: (r) => r["_measurement"] == "chart_event")
|> filter(fn: (r) => r["subject_id"] == "191")
|> filter(fn: (r) => r["_field"] == "value" or r["_field"]
   == "icustay_id" or r["_field"] == "hadm_id")
|> pivot(rowKey: ["_time"], columnKey: ["_field"],
   valueColumn: "_value")
```

Sorgu 33: Bir Sepsis Hastasının belirli bir saat aralığındaki tüm yatış çizelgesi ölçüt değerlerini elde eden Flux sorgusu

5.2.2.2 Anahtar Değer Tekil Sorgular

Bu bölümde de önceki bölümle benzer olarak çoklu depo ortamında anahtar-değer olarak Redis'te saklanan veriler için oluşturulan sorgular daha sonra kıyaslanmak üzere SQL eşlenikleri ile birlikte listelenmektedir.

Nabız (Heart Rate) ölçütünün tüm özelliklerini ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-34'te ve anahtar-değer veritabanı Redis'ten elde eden anahtar-değer sorgusu ise Sorgu-35'te görülmektedir.

```
SELECT * FROM mimiciii.d_items WHERE itemid = 211
```

Sorgu 34: "Heart Rate" ölçütünün tüm özelliklerini listeleyen SQL sorgusu

```
lrange 211 0 -1
```

Sorgu 35: "Heart Rate" ölçütünün tüm özelliklerini listeleyen Redis sorgusu

Solunum Hızı (Respiratory Rate) ölçütünün tüm özelliklerini ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-36'da ve anahtar-değer veritabanı Redis'ten elde eden anahtar-değer sorgusu ise Sorgu-37'de görülmektedir.

```
SELECT * FROM mimiciii.d_items WHERE itemid = 618
```

Sorgu 36: "Respiratory Rate" ölçütünün tüm özelliklerini listeleyen SQL sorgusu

```
lrange 618 0 -1
```

Sorgu 37: "Respiratory Rate" ölçütünün tüm özelliklerini listeleyen Redis sorgusu

Oksijen Satürasyonu (SpO2) ölçütünün tüm özelliklerini ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-38'de ve anahtar-değer veritabanı Redis'ten elde eden anahtar-değer sorgusu ise Sorgu-39'da görülmektedir.

```
SELECT * FROM mimiciii.d_items WHERE itemid = 646
```

Sorgu 38: "SpO2" ölçütünün tüm özelliklerini listeleyen SQL sorgusu

```
lrange 646 0 -1
```

Sorgu 39: "SpO2" ölçütünün tüm özelliklerini listeleyen Redis sorgusu

Fahrenheit cinsinden vücut sıcaklığı (Temperature F) ölçütünün tüm özelliklerini ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-40'ta ve anahtar-değer veritabanı Redis'ten elde eden anahtar-değer sorgusu ise Sorgu-41'de görülmektedir.

```
SELECT * FROM mimiciii.d_items WHERE itemid = 678
```

Sorgu 40: "Temperature F" ölçütünün tüm özelliklerini listeleyen SQL sorgusu

```
lrange 678 0 -1
```

Sorgu 41: "Temperature F" ölçütünün tüm özelliklerini listeleyen Redis sorgusu

Glasgow Koma Ölçeği Toplamı (GCS Total) ölçütünün tüm özelliklerini ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-42'de ve anahtar-değer veritabanı Redis'ten elde eden anahtar-değer sorgusu ise Sorgu-43'te görülmektedir.

```
SELECT * FROM mimiciii.d_items WHERE itemid = 198
```

Sorgu 42: "GCS Total" ölçütünün tüm özelliklerini listeleyen SQL sorgusu

```
lrange 198 0 -1
```

Sorgu 43: "GCS Total" ölçütünün tüm özelliklerini listeleyen Redis sorgusu

Nabız (Heart Rate), Solunum Hızı (Respiratory Rate), Oksijen Satürasyonu (SpO2), Fahrenheit cinsinden vücut sıcaklığı (Temperature F) ve Glasgow Koma Ölçeği Toplamı (GCS Total) ölçütlerinin tüm özelliklerinin ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-44'te ve anahtar-değer veritabanı Redis'ten elde eden anahtar-değer sorgusu ise Sorgu-45'te görülmektedir.

```
SELECT * FROM mimiciii.d_items WHERE itemid IN (211, 618,
    646, 678, 198)
```

Sorgu 44: "Heart Rate", "Respiratory Rate", "SpO2", "Temperature F" ve "GCS Total" ölçütünün tüm özelliklerini listeleyen SQL sorgusu

```
MULTI
LRANGE 211 0 -1
LRANGE 618 0 -1
LRANGE 646 0 -1
LRANGE 678 0 -1
LRANGE 198 0 -1
EXEC
```

Sorgu 45: "Heart Rate", "Respiratory Rate", "SpO2", "Temperature F" ve "GCS Total" ölçütünün tüm özelliklerini listeleyen Redis sorgusu

Bir sepsis hastasına belirli iki hastane kabulü (hospital admission) sırasında uygulanan tedavilerin ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-46'da ve anahtar-değer veritabanı Redis'ten elde eden anahtar-değer sorgusu ise Sorgu-47'de görülmektedir.

```
SELECT * FROM mimiciii.procedures_icd WHERE subject_id=191
    AND (hadm_id=142081 OR hadm_id=136614)
```

Sorgu 46: Bir sepsis hastasına belirli iki hastane kabulü (hospital admission) boyunca uygulanan tedavileri listeleyen SQL sorgusu

```
ZRANGE 191-142081 0 -1 WITHSCORES
```

```
ZRANGE 191-136614 0 -1 WITHSCORES
```

Sorgu 47: Bir sepsis hastasına belirli iki hastane kabulü (hospital admission) boyunca uygulanan tedavileri listeleyen Redis sorgusu

Bir sepsis hastasına belirli iki hastane kabulü (hospital admission) sırasında konulan tanıları ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-48'de ve anahtar-değer veritabanı Redis'ten elde eden anahtar-değer sorgusu ise Sorgu-49'da görülmektedir.

```
SELECT * FROM mimiciii.diagnoses_icd WHERE subject_id=191
    AND (hadm_id=142081 OR hadm_id=136614)
```

Sorgu 48: Bir sepsis hastasına belirli iki hastane kabulü (hospital admission) boyunca konulan tanıları listeleyen SQL sorgusu

```
ZRANGE 191-142081 0 -1 WITHSCORES
ZRANGE 191-136614 0 -1 WITHSCORES
```

Sorgu 49: Bir sepsis hastasına belirli iki hastane kabulü (hospital admission) boyunca konulan tanıları listeleyen Redis sorgusu

Etiket değeri (label) "Magnesium" olan kalemleri (items) ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-50'de ve anahtar-değer veritabanı Redis'ten elde eden anahtar-değer sorgusu ise Sorgu-51'de görülmektedir.

```
SELECT * FROM mimiciii.d_items WHERE label LIKE 'Magnesium'
```

Sorgu 50: Etiket değeri (label) "Magnesium" olan kalemleri (items) listeleyen SQL sorgusu

```
KEYS 'Magnesium'
LRANGE 'Magnesium' 0 -1
```

Sorgu 51: Etiket değeri (label) "Magnesium" olan kalemleri (items) listeleyen Redis sorgusu

Etiket değeri (label) "Arterial Blood Pressure" ile başlayan kalemleri (items) ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-52'de

ve anahtar-değer veritabanı Redis'ten elde eden anahtar-değer sorgusu ise Sorgu-53'te görülmektedir.

```
SELECT * FROM mimiciii.d_items WHERE label LIKE 'Arterial
    Blood Pressure%'
```

Sorgu 52: Etiket değeri (label) "Arterial Blood Pressure" ile başlayan kalemleri (items) listeleyen SQL sorgusu

```
KEYS 'Arterial Blood Pressure*'
LRANGE 'Arterial Blood Pressure diastolic' 0 -1
LRANGE 'Arterial Blood Pressure Alarm - High' 0 -1
LRANGE 'Arterial Blood Pressure systolic' 0 -1
LRANGE 'Arterial Blood Pressure mean' 0 -1
LRANGE 'Arterial Blood Pressure Alarm - Low' 0 -1
```

Sorgu 53: Etiket değeri (label) "Arterial Blood Pressure" ile başlayan kalemleri (items) listeleyen Redis sorgusu

5.2.2.3 Arama-Belge Tekil Sorgular

Bu bölümde çoklu depo ortamında arama-belge odaklı olarak Elasticsearch'te saklanan veriler için oluşturulan sorgular daha sonra kıyaslanmak üzere SQL eşlenikleri ile birlikte listelenmektedir.

İçerisinde "hypersplenism" geçen tüm notlandırmaları (metin belgelerini) ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-54'te ve arama-belge veritabanı Elasticsearch'ten elde eden JSON sorgusu ise Sorgu-55'te görülmektedir.

```
SELECT * FROM mimiciii.noteevents WHERE LOWER(text) LIKE '%
    hypersplenism%'
```

Sorgu 54: İçerisinde "hypersplenism" geçen tüm notlandırmaları (metin belgelerini) listeleyen SQL sorgusu

```
{
    "bool" : {
```

```
      "must" : [
        {
            "query_string": {
              "query": "Hypersplenism",
              "default_field": "text"
            }
        }
      ]
    }
},
"size":300
```

Sorgu 55: İçerisinde "hypersplenism" geçen tüm notlandırmaları (metin belgelerini) listeleyen Elasticsearch JSON sorgusu

İçerisinde "glomerulonephritis" geçen tüm notlandırmaları (metin belgelerini) ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-56'da ve arama-belge veritabanı Elasticsearch'ten elde eden JSON sorgusu ise Sorgu-57'de görülmektedir.

```
SELECT * FROM mimiciii.noteevents WHERE LOWER(text) LIKE '%
    glomerulonephritis%'
```

Sorgu 56: İçerisinde "glomerulonephritis" geçen tüm notlandırmaları (metin belgelerini) listeleyen SQL sorgusu

```
{
    "bool" : {
      "must" : [
        {
            "query_string": {
              "query": "glomerulonephritis",
              "default_field": "text"
            }
        }
      ]
```

```
    }
},
"size":1000
```

Sorgu 57: İçerisinde "glomerulonephritis" geçen tüm notlandırmaları (metin belgelerini) listeleyen Elasticsearch JSON sorgusu

İçerisinde "glomerulonephritis" geçen ve taburcu notları "Discharge summary" kategorisindeki tüm notlandırmaları (metin belgelerini) ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-58'de ve arama-belge veritabanı Elasticsearch'ten elde eden JSON sorgusu ise Sorgu-59'da görülmektedir.

```
SELECT * FROM mimiciii.noteevents WHERE category='Discharge
    summary' AND LOWER(text) LIKE '%glomerulonephritis%'
```

Sorgu 58: İçerisinde "glomerulonephritis" geçen ve taburcu notları "Discharge summary" kategorisindeki tüm notlandırmaları (metin belgelerini) listeleyen SQL sorgusu

```
{
"bool" : {
      "must" : [
   { "term" : { "category": "Discharge summary" } },
        {
            "query_string": {
              "query": "glomerulonephritis",
              "default_field": "text"
            }
        }
      ]
    }
},
```

```
"size":1000
```

Sorgu 59: İçerisinde "glomerulonephritis" geçen ve taburcu notları "Discharge summary" kategorisindeki tüm notlandırmaları (metin belgelerini) listeleyen Elasticsearch JSON sorgusu

Belirli bir sepsis hastasının içerisinde kan basıncı "blood pressure" geçen tüm notlandırmalarını (metin belgelerini) ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-60'ta ve arama-belge veritabanı Elasticsearch'ten elde eden JSON sorgusu ise Sorgu-61'de görülmektedir.

```
SELECT * FROM mimiciii.noteevents WHERE subject_id=191 AND
   LOWER(text) LIKE '%blood pressure%'
```

Sorgu 60: Belirli bir sepsis hastasının içerisinde kan basıncı "blood pressure" geçen tüm notlandırmalarını (metin belgelerini) listeleyen SQL sorgusu

```
{
"bool" : {
      "must" : [
     { "term" : { "subject_id": "191" } },
        {
            "query_string": {
              "query": "blood pressure",
              "default_field": "text"
            }
        }
      ]
    }
},
"size":1000
```

Sorgu 61: Belirli bir sepsis hastasının içerisinde kan basıncı "blood pressure" geçen tüm notlandırmalarını (metin belgelerini) listeleyen Elasticsearch JSON sorgusu

Belirli bir sepsis hastasının içerisinde kan basıncı "blood pressure" geçen ve taburcu notları "Discharge summary" kategorisindeki tüm notlandırmalarını (metin belgelerini) ilişkisel veritabanı Postgresql'den elde eden SQL sorgusu Sorgu-62'de ve arama-belge veritabanı Elasticsearch'ten elde eden JSON sorgusu ise Sorgu-63'te görülmektedir.

```
SELECT * FROM mimiciii.noteevents WHERE subject_id=191 AND
    category='Discharge summary' AND LOWER(text) LIKE '%blood
     pressure%'
```

Sorgu 62: Belirli bir sepsis hastasının içerisinde kan basıncı "blood pressure" geçen ve taburcu notları "Discharge summary" kategorisindeki tüm notlandırmalarını (metin belgelerini) listeleyen SQL sorgusu

```
{
"bool" : {
      "must" : [
     { "term" : { "subject_id": "191" } },
       { "term" : { "category": "Discharge summary" } },
        {
            "query_string": {
              "query": "blood pressure",
              "default_field": "text"
            }
        }
      ]
    }
},
"size":1000
```

Sorgu 63: Belirli bir sepsis hastasının içerisinde kan basıncı "blood pressure" geçen ve taburcu notları "Discharge summary" kategorisindeki tüm notlandırmalarını (metin belgelerini) listeleyen Elasticsearch JSON sorgusu

5.2.2.4 Çoklu Depo Birleşik Sorguları

Bu bölümde çoklu depo ortamının önemini daha da ön plana çıkaracak olan birleşik sorgular listelenmektedir. Bölüm-4.1.5'teki kullanım durumları ile bağlantılı ve temel olarak sepsis tespiti, ölüm oranı tahminleme ve klinik metin sınıflandırmaya yönelik sorgular oluşturulması amaçlanmıştır. Bileşik sorgular oluşturulurken Bölüm-5.2.2'deki tekil sorgulardan faydalanılarken, farklı depolardan sorgulamayı barındıran yeni sorgu kombinasyonları da kullanılmıştır.

Belirli bir hastanın 1532 tanımlayıcılı (id) kalem (item) değerine sahip metrik değerlerini bu metriğin etiketi (Magnezyum - "Magnesium") ile birlikte ilişkisel veritabanı Postgresql'den elde eden SQL Sorgusu Sorgu-64'te ve zaman serisi veritabanı InfluxDB ile anahtar değer veritabanı Redis'ten elde eden çoklu depo (polystore) sorgusu ise Sorgu-65'te görülmektedir.

```
SELECT chartevents.*,d_items.label FROM
mimiciii.chartevents INNER JOIN mimiciii.d_items
ON chartevents.itemid = d_items.itemid
WHERE chartevents.itemid=1532 AND chartevents.subject_id=21
```

Sorgu 64: Belirli bir hastanın 1532 tanımlayıcılı (id) kalem (item) değerine sahip metrik değerlerini bu metriğin etiketi ile birlikte listeleyen SQL sorgusu

```
----------
*Redis*
----------
[0] [lrange] [{1532}]

----------
*InfluxDB*
----------
from(bucket: "mimic-iii")
|> range(start:2000-01-01, stop:2012-12-31)
|> filter(fn: (r) => r["_measurement"] == "chart_event")
|> filter(fn: (r) => r["itemid"] == "1532")
```

```
|> filter(fn: (r) => r["subject_id"] == "21")
|> filter(fn: (r) => r["_field"] == "value" or r["_field"]
   == "icustay_id" or r["_field"] == "hadm_id")
|> pivot(rowKey: ["_time"], columnKey: ["_field"],
   valueColumn: "_value")
```

Sorgu 65: Belirli bir hastanın 1532 tanımlayıcılı (id) kalem (item) değerine sahip metrik değerlerini bu metriğin etiketi ile birlikte listeleyen Polystore (Redis $\bowtie$ $InfluxDB$)$sorgusu$

Notlarında "azotemia" geçen hastalardan "Sepsis" teşhisi konanları ilişkisel veritabanı Postgresql'den elde eden SQL Sorgusu Sorgu-66'da ve zaman serisi veritabanı InfluxDB ile anahtar değer veritabanı Redis'ten elde eden çoklu depo (polystore) sorgusu ise Sorgu-67'de görülmektedir.

```
SELECT * FROM mimiciii.noteevents
INNER JOIN mimiciii.diagnoses_icd ON noteevents.subject_id=
   diagnoses_icd.subject_id
WHERE LOWER(noteevents.text) LIKE '%azotemia%' AND
   diagnoses_icd.icd9_code='99591'
```

Sorgu 66: Notlarında "azotemia" geçen hastalardan "Sepsis" teşhisi konanları listeleyen SQL sorgusu

```
--------
*Redis*
--------
[6] [reverselrange] [{99591}]

------------------
*Elasticsearch*
------------------
{
"bool" : {
        "must" : [
        {
```

```
            "query_string": {
              "query": "azotemia",
              "default_field": "text"
            }
          }
        ]
      }
},
"size":10000
```

Sorgu 67: Notlarında "azotemia" geçen hastalardan "Sepsis" teşhisi konanları listeleyen Polystore (Redis ⋈ *Elasticsearch*)*sorgusu*

Notlarında "azotemia" geçen hastalardan "Sepsis" veya "Severe Sepsis" teşhisi konanları ilişkisel veritabanı Postgresql'den elde eden SQL Sorgusu Sorgu-68'de ve zaman serisi veritabanı InfluxDB ile anahtar değer veritabanı Redis'ten elde eden çoklu depo (polystore) sorgusu ise Sorgu-69'da görülmektedir.

```
SELECT * FROM mimiciii.noteevents
INNER JOIN mimiciii.diagnoses_icd ON noteevents.subject_id=
    diagnoses_icd.subject_id
WHERE LOWER(noteevents.text) LIKE '%azotemia%' AND (
    diagnoses_icd.icd9_code IN('99591','99592'))
```

Sorgu 68: Notlarında "azotemia" geçen hastalardan "Sepsis" veya "Severe Sepsis" teşhisi konanları listeleyen SQL sorgusu

```
--------
*Redis*
--------
[6] [reverselrange] [{99591}{99592}]

------------------
*Elasticsearch*
------------------
```

```
{
"bool" : {
        "must" : [
        {
            "query_string": {
              "query": "azotemia",
              "default_field": "text"
            }
        }
      ]
    }
},
"size":10000
```

Sorgu 69: Notlarında "azotemia" geçen hastalardan "Sepsis" veya "Severe Sepsis" teşhisi konanları listeleyen Polystore (Redis ⋈ *Elasticsearch*)*sorgusu*

Notlarında "azotemia" geçen hastalardan "Sepsis" veya "Severe Sepsis" teşhisi konanlardan hastanede ölenleri ilişkisel veritabanı Postgresql'den elde eden SQL Sorgusu Sorgu-70'te ve zaman serisi veritabanı InfluxDB ile anahtar değer veritabanı Redis'ten elde eden çoklu depo (polystore) sorgusu ise Sorgu-71'de görülmektedir.

```
SELECT * FROM mimiciii.noteevents
INNER JOIN mimiciii.diagnoses_icd
ON noteevents.subject_id = diagnoses_icd.subject_id
INNER JOIN mimiciii.admissions
ON noteevents.hadm_id = admissions.hadm_id
WHERE LOWER(noteevents.text) LIKE '%azotemia%'
AND (diagnoses_icd.icd9_code IN ('99591', '99592'))
AND admissions.deathtime IS NOT NULL
```

Sorgu 70: Notlarında "azotemia" geçip "Sepsis" veya "Severe Sepsis" teşhisi konan ve hastanede ölenleri listeleyen SQL sorgusu

```
————————
```

```
*Redis*
--------
[6] [reverselrange] [{99591}{99592}]

---------------
*Postgresql*
---------------
SELECT * FROM mimiciii.admissions WHERE admissions.deathtime
    IS NOT NULL

-----------------
*Elasticsearch*
-----------------
{
"bool" : {
        "must" : [
        {
            "query_string": {
              "query": "azotemia",
              "default_field": "text"
            }
        }
      ]
    }
},
"size":10000
```

Sorgu 71: Notlarında "azotemia" geçip "Sepsis" veya "Severe Sepsis" teşhisi konan ve hastanede ölenleri listeleyen Polystore (Redis $\bowtie$ *Postgresql* $\bowtie$ *Elasticsearch*)*sorgusu*

Bir sepsis hastasının kabul verisi ile birlikte ona uygulanan tedavileri ilişkisel veritabanı Postgresql'den elde eden SQL Sorgusu Sorgu-72'de ve ilişkisel veritabanı Postgresql ile anahtar değer veritabanı Redis'ten elde eden

çoklu depo (polystore) sorgusu ise Sorgu-73'te görülmektedir.

```
SELECT * FROM mimiciii.admissions INNER JOIN mimiciii.
    procedures_icd ON admissions.subject_id=procedures_icd.
    subject_id AND admissions.hadm_id = procedures_icd.
    hadm_id WHERE admissions.subject_id=191
```

Sorgu 72: Bir sepsis hastasının kabul verisi ile birlikte ona uygulanan tedavileri listeleyen SQL sorgusu

```
----------
*Redis*
----------
[5] [zrangeWithScore] [{191-136614}{191-142081}]

----------------
*Postgresql*
----------------
SELECT * FROM mimiciii.admissions WHERE subject_id=191
```

Sorgu 73: Bir sepsis hastasının kabul verisi ile birlikte ona uygulanan tedavileri listeleyen Polystore (Redis $\bowtie$ *Postgresql*)*sorgusu*

Bir sepsis hastasına uygulanan prosedürlerle birlikte hastanın Temperature F, yatış ve hastane kabul verisini ilişkisel veritabanı Postgresql'den elde eden SQL Sorgusu Sorgu-74'te ve zaman serisi veritabanı InfluxDB, ilişkisel veritabanı Postgresql ve anahtar değer veritabanı Redis'ten elde eden çoklu depo (polystore) sorgusu ise Sorgu-75'te görülmektedir.

```
SELECT * FROM mimiciii.admissions
INNER JOIN mimiciii.procedures_icd ON admissions.subject_id
    = procedures_icd.subject_id
INNER JOIN mimiciii.chartevents ON admissions.subject_id =
    chartevents.subject_id
WHERE admissions.subject_id = 191
```

```
AND chartevents.itemid IN (678)
```

Sorgu 74: Bir sepsis hastasına uygulanan prosedürlerle birlikte hastanın "Temperature F" yatış ve hastane kabul verisini listeleyen SQL sorgusu

```
--------
*Redis*
--------
[5] [zrangeWithScore] [{191-136614}{191-142081}]

----------------
*Postgresql*
----------------
SELECT * FROM mimiciii.admissions WHERE subject_id=191

------------------
*InfluxDB*
------------------
from(bucket: "mimic-iii")
|> range(start:2000-01-01, stop:2012-12-31)
|> filter(fn: (r) => r["_measurement"] == "chart_event")
|> filter(fn: (r) => r["itemid"] == "678")
|> filter(fn: (r) => r["subject_id"] == "191")
|> filter(fn: (r) => r["_field"] == "value" or r["_field"]
   == "icustay_id" or r["_field"] == "hadm_id")
|> pivot(rowKey: ["_time"], columnKey: ["_field"],
   valueColumn: "_value")
```

Sorgu 75: Bir sepsis hastasına uygulanan prosedürlerle birlikte hastanın Temperature F yatış ve hastane kabul verisini listeleyen Polystore (Redis $\bowtie$ *Postgresql* $\bowtie$ *InfluxDB*)*sorgusu*

İçinde "azotemia" geçen notlardaki hastalardan "Acute kidney failure, unspecified" teşhisi konulan ve hastanede ölmüş olanların, hastanede yattıkları belirli tarihler arasındaki respiratory rate'inin ilişkisel veritabanı Postgre-

sql'den elde eden SQL Sorgusu Sorgu-76'da ve zaman serisi veritabanı InfluxDB, ilişkisel veritabanı Postgresql, arama-belge veritabanı Elasticsearch ve anahtar değer veritabanı Redis'ten elde eden çoklu depo (polystore) sorgusu ise Sorgu-77'te görülmektedir.

```
SELECT * FROM mimiciii.noteevents
INNER JOIN mimiciii.diagnoses_icd ON noteevents.subject_id =
    diagnoses_icd.subject_id
INNER JOIN mimiciii.admissions ON noteevents.hadm_id =
    admissions.hadm_id
INNER JOIN mimiciii.chartevents ON admissions.subject_id =
    chartevents.subject_id
WHERE LOWER(noteevents.text) LIKE '%azotemia%'
AND (diagnoses_icd.icd9_code = '5849')
AND admissions.deathtime IS NOT NULL
AND chartevents.itemid = 618
AND chartevents.charttime BETWEEN '2180-10-01' AND '
    2189-12-31';
```

Sorgu 76: İçinde "azotemia" geçen notlardaki hastalardan "Acute kidney failure, unspecified" teşhisi konulan ve hastanede ölmüş olanların, hastanede yattıkları belirli tarihler arasındaki respiratory rate'ini listeleyen SQL sorgusu

```
--------
*Redis*
--------
[6] [reverselrange] [{5849}]

----------------
*Postgresql*
----------------
SELECT * FROM mimiciii.admissions WHERE admissions.deathtime
    IS NOT NULL

------------------
```

```
*Elasticsearch*
-----------------
{
"bool" : {
        "must" : [
        {
            "query_string": {
              "query": "azotemia",
              "default_field": "text"
            }
        }
      ]
    }
},
"size":10000

-----------------
*InfluxDB*
-----------------
from(bucket: "mimic-iii")
|> range(start:2009-10-28T00:00:00Z, stop:2009-12-10T00
   :00:00Z)
|> filter(fn: (r) => r["_measurement"] == "chart_event")
|> filter(fn: (r) => r["itemid"] == "618")
|> filter(fn: (r) => r["_field"] == "value" or r["_field"]
   == "icustay_id" or r["_field"] == "hadm_id")
|> pivot(rowKey: ["_time"], columnKey: ["_field"],
   valueColumn: "_value")
```

Sorgu 77: İçinde “azotemia” geçen notlardaki hastalardan “Acute kidney failure, unspecified” teşhisi konulan ve hastanede ölmüş olanların, hastanede yattıkları belirli tarihler arasındaki respiratory rate’ini listeleyen Polystore (Redis $\bowtie$ *Postgresql* $\bowtie$ *Elasticsearch* $\bowtie$ *InfluxDB*)*sorgusu*

5.3 Sorgu Sonuçları ve Çözümleme

Bölüm-5.2.2'de tanımlanan sorgular hem yerel hem de küme bilgisayar yapılandırması üzerinde çalışabilecek durumdadır ve her iki yapılandırma üzerinde de test edilmiştir. Elde edilen sonuçların tamamı yerel yapılandırma üzerinde alınmış olup, bunun başlıca sebebi ise küme yapılandırmasındaki bilgisayarlara erişilebilirliğin her zaman mümkün olmaması ve bu durumun tahmin edilemez olmasıdır. Değerlendirme, sorgu kümesinin işletimi ile gerçekleştirildiğinden, sorguların işletim sürelerine bağımlıdır. Tüm sorguların işletiminin tamamlanması da zaman aldığından, değerlendirme ortamının durağan (stabil) olması ve bu ortama hızlı erişilebilmesi esastır. Ayrıca yapılandırmaları kıyaslama açısından küme bilgisayar ortamı üzerinde elde edilen sonuçlar ile yerel yapılandırma üzerinde elde edilen sonuçlar birbirine oldukça yakındır. Dolayısıyla sorgu kümesinin işletimi ile değerlendirme, aynı zamanda geliştirim yapılan yerel bilgisayar olan M1 çip, 16 GB Ram, 500 GB SSD yapılandırmasına sahip Macbook Air üzerinde gerçekleştirilmiştir.

İlerleyen kısımlarda öncelikle Bölüm-5.3.1'de InfluxDB, Redis ve Elasticsearch üzerinde çalıştırılan tekil sorgu sonuçları incelenecektir. Bölüm-5.3.2'de ise çoklu depo ortamındaki kullanım durumları ile bağlantılı birleşik sorgu sonuçları listelenmektedir. Tekil ve birleşik sorgu sonuçları üzerindeki her iki değerlendirmede de kıyas noktası olarak ilişkisel depo seçilmiş olup Postgresql ile performans kıyaslaması yapılmıştır. Postgresql üzerinde iki farklı veri yapılandırması mevcuttur. İlkinde kıyaslama amacıyla MIMIC-III veri kümesinin tamamı Postgresql üzerinde "mimic" isimli şemaya, diğerinde ise çoklu depo ortamının ilişkisel bileşeni olarak MIMIC-III veri kümesinin belirlenen alt parçaları "poly_mimic_rel" isimli şemaya yüklenmiştir. Kıyaslama noktasında ilişkisel model, her çeşit ve yapıda verinin ve içerdiği ilişkilerin saklanabileceği genel geçer çözüm sağlayabilen en iyi, köklü ve olgun mimarilerden biri olduğu, Postgresql de ilişkisel model üzerine kurulmuş en gelişmiş ve performanslı veritabanı çözümlerinden biri olduğu için tercih edilmiştir. Ayrıca MIMIC-III kod

tabanında[55] da bahsedildiği üzere, verinin modelleme yaklaşımı olarak ilişkisel model ve saklama yaklaşımı olarak da Postgresql tercih edilmiştir. Bu durum Postgresql'in verinin tek bir model ve veritabanı ile temsil edilebileceği en uygun çözümlerden biri olduğunu dolayısıyla da kıyaslama açısından seçildiğini göstermektedir. Çoklu depo sistemleri büyük veri saklama ve sorgulama bakış açılarından bir eniyileştirme yaklaşımıdır. Dolayısıyla kritik noktalarda uygun modelleme yaklaşımları ile teknoloji harikası saklama çözümleri kullanılmış ve geliştirilen etkin birleşim algoritması ile de performans bakımından üstün sonuçlar elde edilmiştir.

5.3.1 Tekil Sorgu Sonuçları

Bu bölümde, Bölüm-5.2.2'de kullanım durumu analizi sonucu ortaya çıkarılan metrikler üzerinden oluşturulan basit tekil sorgular ile çoklu deponun bileşenleri sınanmıştır. Postgresql kıyaslama noktası olduğundan AQuAPooL çoklu depo sisteminin dağıtık veri ortamında bulunan her bir veri modeline ait deponun sorgusu ile karşılaştırılmıştır. Elde edilen sonuçlara göre; tekil sorguların doğrudan yerel depoların sorgu arayüzleri aracılığıyla çalıştırılması ile AQuAPooL üzerinden çalıştırılması arasında işletim süreleri bakmından en fazla %5 oranında bir gecikme farkı ortaya çıkmaktadır. Bunun başlıca nedeni sonuçların iletilmesi sırasında ortaya çıkan ağ maliyetidir ve sonuç boyutuna göre değişmektedir.

5.3.1.1 Zaman Serisi Sorgu Sonuçları

Öncelikle Bölüm-5.2.2.1'de ortaya konan zaman serisi sorguları AQuAPooL aracılığyla InfluxDB ve Postgresql üzerinde çalıştırılarak sonuçlar Tablo-5.2'deki gibi elde edilmiştir. Buradaki sorgularda hastaların CHARTEVENTS tablosunda bulunan yatış verileri incelenerek, sepsis tespiti ve ölüm oranı tahminleme için kritik öneme sahip metriklerden bazıları (arter kan basıncı, nabız, solunum hızı, oksijen satürasyonu, vücut sıcaklığı, glasgow koma

[55] https://github.com/MIT-LCP/mimic-code/tree/main/mimic-iii

ölçeği) hem standart, hem de zaman aralıklı olarak sorgulanarak performans çözümlemesi yapılmaktadır. İlk olarak tüm yatış yapan hastalar arasında arter kan basıncı (arterial blood pressure) değerinin belirli tarih aralığında Postgresql üzerinden sorgulanması Sorgu-8'de, InfluxDB üzerinden sorgulanması ise Sorgu-9'da gerçekleştirilmiştir. Sorgu Postgresql üzerinde 4.165 saniye, InfluxDB'de ise 0.4 saniye sürmüştür ve tarih aralığı içerdiğinden aradaki performans farkı daha yüksek olmuştur.

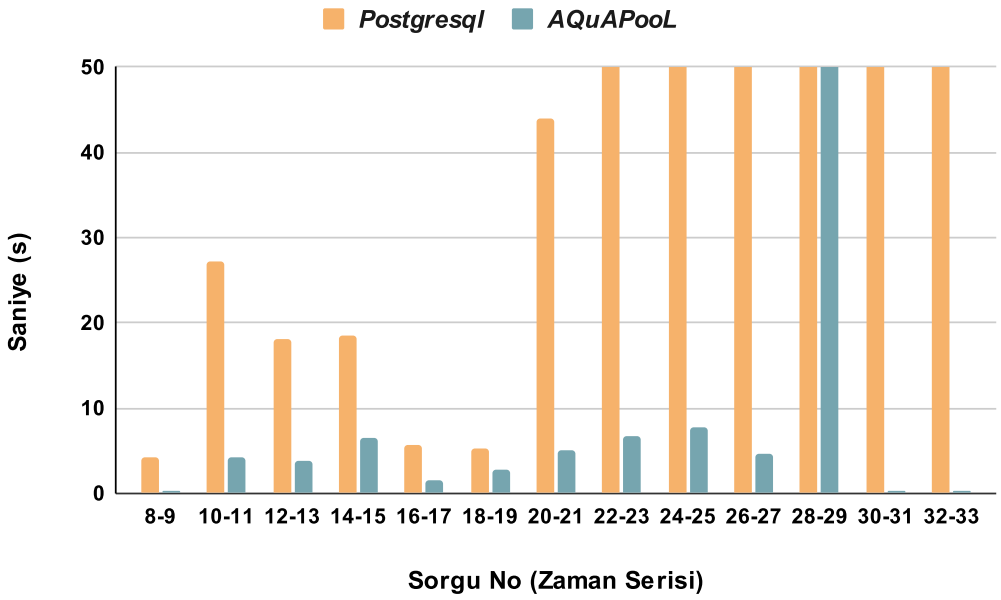
Bunun dışında sepsis teşhisi konulan bir hastanın sırasıyla "Heart Rate", "Respiratory Rate", "SpO2", "Temperature F", "Glasgow Coma Scale Total" metrikleri Sorgu-10, 12, 14, 16, 18 SQL sorguları ile Postgresql üzerinden, Sorgu-11, 13, 15, 17, 19 Flux sorguları ile InfluxDB üzerinden sorgulanmıştır. Sonuçlardan anlaşılacağı üzere sorgular InfluxDB ile 2 - 4.5 kat arasında daha performanslı işletilmiştir.

Ayrıca belirlenen bu 5 metriğin kademeli olarak ikili, üçlü, dörtlü ve beşli kombinasyonda birleştirilerek Sorgu-20, 22, 24, 26 SQL sorguları ile Postgresql'den, Sorgu-21, 23, 25, 27 Flux sorguları ile InfluxDB üzerinden sorgulanmıştır. Sonuçlara göre InfluxDB'nin sorgu işletim performansının 9 - 16 kat daha yüksek olduğu ortaya konmuştur. Böylece sorgulanan kayıt sayısı arttıkça zaman aralığı olmayan sorgularda dahi performans farkı daha net bir şekilde görülmektedir.

Son olarak her iki veritabanının da performans sınırlarını zorlayan daha kapsamlı üç tip sorgu işletimi yapılmıştır. Buna göre sepsis teşhisi konan bir hastanın tüm metrikleri Sorgu-28 SQL sorgusu ile Postgresql, Sorgu-29 Flux sorgusu ile InfluxDB üzerinden elde edilmektedir. Buna göre 330 milyon kayıt arasından, seçilen bir hastanın tüm verileri elde edilmekte olduğundan, oldukça büyük bir veri bloğu sorgulanmaktadır ve Postgresql sorgu işletimi 2017 saniye sürmüşken, InfluxDB sorgu işletimi de yaklaşık 73 saniye sürmüştür. Dolayısıyla seçiciliği düşük olan bu sorgu ile hem sorgu işletimi hem de sonuçların elde edilmesi büyük zaman almaktadır. Kıyaslama yapıldığında, InfluxDB sorguyu Postgresql'e göre yaklaşık 27.5 kat daha hızlı çalıştırmıştır. Benzer şekilde belirli bir sepsis hastasının verisi bu sefer farklı

zaman aralıklarında ancak aynı süre boyunca filtrelenerek Sorgu-30 ve 32'da SQL sorgusu olarak Postgresql'de, Sorgu-31 ve 33'de Flux sorgusu olarak InfluxDB'de sorgulanmıştır. Sorgu performansı Postgresql üzerinde 1000-1100 saniye aralığında elde edilmiş olup, InfluxDB'de ise 0.5 saniyenin altında ortaya çıkmıştır. Dolayısıyla InfluxDB bu iki sorgu için Postgresql'e göre 3333 - 4056 kat daha yüksek performans sergilemiştir. Zaman serisi indekse sahip olması sayesinde, zaman aralığındaki sonuçların filtrelemesini çok hızlı gerçekleştirerek sonuç kümesini hızlıca daralttığından InfluxDB'nin gerçek farkı ve avantajı burada gözler önüne serilmiştir.

Anlatılan tüm sorguların Postgresql ve InfluxDB üzerinden kıyaslamalı çözümlemesi Şekil-5.1'de grafiksel olarak ortaya konmuştur. Tablo-5.2'den görüldüğü üzere sorgu sonuçları için zaman ölçeği oldukça geniş olduğundan, kıyaslamadaki görselliği bozmamak ve farkedilebilirliği artırmak adına adına grafik üzerindeki zaman limiti 50 saniye olarak belirlenmiştir.



Şekil 5.1: Postgresql vs AQuAPooL Zaman Serisi Tekil Sorgu Sonucu Kıyaslaması (s cinsinden)

Sorgu Tanımı	Postgresql Sorgu No	Postgresql İşletimi (s)	AQuAPooL Sorgu No	AQuAPooL İşletimi (s)	Sonuç Boyutu (satır)
Tüm Hastaların İki Senelik ABP Değerleri	8	4.16	9	0.40	46966
Bir Sepsis Hastasının Heart Rate Değerleri	10	27.30	11	4.26	284
Bir Sepsis Hastasının Respiratory Rate Değerleri	12	18.03	13	3.73	229
Bir Sepsis Hastasının SpO2 Değerleri	14	18.47	15	6.55	285
Bir Sepsis Hastasının Temperature F Değerleri	16	5.62	17	1.46	71
Bir Sepsis Hastasının Glasgow Coma Scale Total Değerleri	18	5.20	19	2.75	66
Bir Sepsis Hastasının Heart Rate ve Respiratory Rate Değerleri	20	44.00	21	4.97	513
Bir Sepsis Hastasının Heart Rate, Respiratory Rate ve SpO2 Değerleri	22	61.00	23	6.66	798
Bir Sepsis Hastasının Heart Rate, Respiratory Rate, SpO2 ve Temperature F Değerleri	24	84.01	25	7.71	869
Bir Sepsis Hastasının Heart Rate, Respiratory Rate, SpO2, Temperature F ve Glasgow Coma Scale Total Değerleri	26	73.00	27	4.56	935
Bir Sepsis Hastasının Tüm Ölçüm Değerleri	28	2017.82	29	72.93	14000
Bir Sepsis Hastasının 08:00-12:00 Aralığındaki 4 Saatlik Tüm Ölçüm Değerleri	30	1014.14	31	0.25	284
Bir Sepsis Hastasının 11:00-15:00 Aralığındaki 4 Saatlik Tüm Ölçüm Değerleri	32	1099.99	33	0.33	394

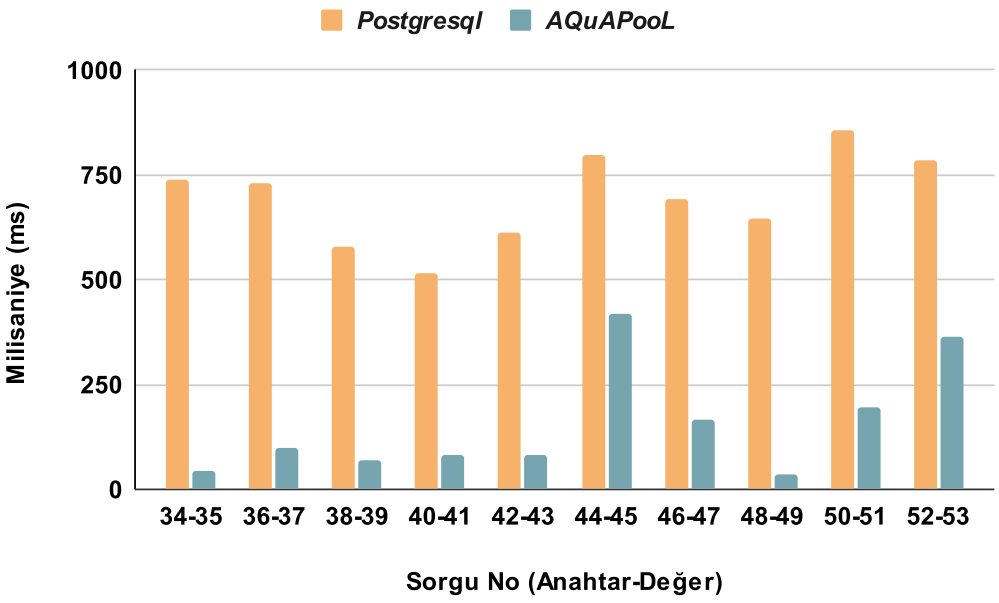
Tablo 5.2: Postgresql ve AQuAPooL Zaman Serisi Tekil Sorgu Sonuçları (s cinsinden)

5.3.1.2 Anahtar Değer Sorgu Sonuçları

İkinci olarak Bölüm-5.2.2.2'te ortaya konan anahtar-değer sorguları Redis ve Postgresql üzerinde çalıştırılarak sonuçlar Tablo-5.3'deki gibi elde edilmiştir. Tüm sorguların Postgresql ve Redis üzerinden kıyaslamalı grafiksel çözümlemesi de Şekil-5.2'de ortaya konmuştur. Bölüm-5.3.1.1'de sorgulama amaçlı kullanılan "Heart Rate", "Respiratory Rate", "SpO2", "Temperature F" ve "Glasgow Coma Scale Total" ölçütlerinin detay bilgilerini içeren D_ITEMS verisi Sorgu-34, 36, 38, 40, 42 SQL sorguları ile Postgresql üzerinden, Sorgu-35, 37, 39, 41, 43 anahtar-değer sorguları ile Redis üzerinden elde edilmiştir. Bu sorguların sonuçlarına göre Postgresql'in sorgu işletim performansı 500 milisaniyenin üzerinde, Redis'in sorgu işletim performansı ise 100 milisaniyenin altındadır. Tablodaki sonuçlara göre Redis'in bu sorguları 6 - 17 kat daha performanslı çalıştırmış olduğu anlaşılmaktadır. Sorgu-44'te Postgresql, Sorgu-45'te ise Redis üzerinden bu metriklerin tümünün değerleri aynı anda elde edilmektedir. Sonuca göre Postgresql sorgusu 796 milisaniye, toplu Redis sorgusu ise 418 milisaniye sürmüştür. Bunun başlıca sebebi tüm metriklerin değerlerini elde etmek için sorgunun Postgresql'de 1 sefer, Redis'te ise 5 sefer işletilmesidir. Dolayısıyla Redis bir çoklu komut sorgusu şeklinde işletildiğinden atılan istek sayısından ötürü Redis'in toplam sorgu süresi artarken, Postgresql'in sabit kalmaktadır. Bahsi geçen bu sorgular aralık (range) sorguları olup verilen anahtara karşılık gelen değerler kümesinin baştan itibaren (başlangıç indeksi: 0) tamamını (bitiş indeksi: -1) sorgulamaktadır.

Sepsis teşhisi konan belirli bir hastaya hastane başvuruları (admissions) boyunca uygulanan tedaviler ve konulan tanılar sıralı bir şekilde Sorgu-46 ve Sorgu-48 ile Postgresql'den, Sorgu-47 ve Sorgu-49 ile Redis'ten sorgulanmaktadır. Sonuçlara göre Postgresql sorgusu önceki sorgu sonuçlarına benzer olarak 650-700 milisaniye arası, Redis sorgusu da 35-170 milisaniye arası sürmüştür ve Redis'in daha burada da daha iyi sonuçlar verdiği görülmektedir. Bahsi geçen sorgular sıralı aralık sorguları olup, sıralı kümeler üzerinde işlem yapmak için eniyileştirilmiştir (optimized).

Son olarak belirli anahtar kelime değerindeki ve bu kelime ile başlayan kalemlerin özelliklerinin listelenmesi işlemi de Sorgu-50 ve Sorgu-52 ile Postgresql'de, Sorgu-51 ve Sorgu-53 ile Redis'te gerçekleştirilmiştir. Sorgu sonuçları Postgresql üzerinde 750-850 milisaniye aralığında gerçekleşmiş olup, Sorgu-51 Redis'te yaklaşık 200 ms, Sorgu-53 ise 365 ms sürmüştür. Burada redis anahtar komutu (keys) kullanılarak öncelikle girilen metin değerini içeren anahtarlar listelenerek, daha sonra bu anahtarlar üzerinden aralık sorgulaması yapılmaktadır. Sorgu-51'te 1 adet anahtar döndüğünden toplam 2 adımda sorgulama yapılmakta, Sorgu-53 ise içerme sorgusu olduğundan ve dolayısıyla da 5 anahtar değeri döndüğünden toplam 6 adımda sorgulanmaktadır. Bu nedenle belirli bir değeri içerme sorgusu istek sayısının fazla olmasından dolayı Redis'te daha uzun sürmüştür. Ancak buna rağmen her iki sorgu da Redis ile Postgresql'e göre daha performanslı sorgulanmıştır.



Şekil 5.2: Postgresql vs AQuAPooL Anahtar Değer Tekil Sorgu Sonucu Kıyaslaması (ms)

Sorgu Tanımı	Postgresql Sorgu No	Postgresql İşletimi (s)	AQuAPooL Sorgu No	AQuAPooL İşletimi (s)	Sonuç Boyutu (satır)
Heart Rate Kaleminin Tüm Özellik Değerleri	34	738	35	43	1
Respiratory Rate Kaleminin Tüm Özellik Değerleri	36	730	37	97	1
SpO2 Kaleminin Tüm Özellik Değerleri	38	577	39	71	1
Temperature F Kaleminin Tüm Özellik Değerleri	40	515	41	80	1
Glasgow Coma Scale Total Kaleminin Tüm Özellik Değerleri	42	611	43	81	1
Heart Rate, Respiratory Rate, SpO2, Temperature F ve Glasgow Coma Scale Total Kalemlerinin Tüm Özellik Değerleri	44	796	45	418	5
Sepsis Teşhisi Konan Bir Hastaya Konulan Tanılar	46	691	47	168	13
Sepsis Teşhisi Konan Bir Hastaya Uygulanan Tedaviler	48	648	49	35	24
'Magnesium' Kelimesini İçeren Kalemler	50	856	51	196	3
'Arterial Blood Pressure' Kelimesi ile Başlayan Kalemler	52	785	53	365	5

Tablo 5.3: Postgresql ve AQuAPooL Anahtar Değer Tekil Sorgu Sonuçları (ms)

5.3.1.3 Arama-Belge Sorgu Sonuçları

Tekil sorgularda son olarak Bölüm-5.2.2.3'te ortaya konan arama-belge sorguları Elasticsearch ve Postgresql üzerinde çalıştırılarak sonuçlar Tablo-5.4'teki gibi elde edilmiştir. Sorguların Postgresql ve Elasticsearch üzerinden kıyaslamalı grafiksel çözümlemesi de Şekil-5.3'de ortaya konmuştur. Hasta notlarında geçen "Hypersplenism" ve "Glomerulonephritis" tanıları ile "blood pressure" ölçütü önce Notevents tablosundaki tüm notlar, daha sonra da taburcu notları içerisinde aranmıştır. Sorgu kombinasyonunu oluşturmadaki

asıl amaç klinik metin sınıflandırması için çözüm olarak tercih edilen arama-belge veritabanı Elasticsearch'ün etkinliğini ortaya çıkararak ispatlamaktır.

Öncelikle Sorgu-54 ile Postgresql'den, Sorgu-55 ile Elasticsearch'ten "Hypersplenism" anahtar kelimesi geçen notlar sorgulanmaktadır. Hipersplenizm, dalağın işini yaparken aşırı aktif hale gelmesidir. Dalağın görevi kan dolaşımındaki eski ve hasar görmüş hücreleri filtrelemektir. Dalak aşırı aktif olmasından dolayı bu filtrelemeyi sürekli yaptığında, çok fazla miktarda kan hücresini erkenden vücuttan atar ve bu da kan hücresi eksikliği olan sitopeni'ye sebep olabilir. Bu sorguya özel olarak ve kullanım durumlarında belirtildiği üzere ICD-9 etiketlemesi amacıyla bir arama yapılması hedeflenmiştir. Sonuçlara göre Postgresql sorgusu 101 saniye, Elasticsearch üzerinde işletilen sorgu ise yaklaşık 2.85 saniye sürmüştür. Buna göre arama uzayı büyük olduğunda ve eşleşen belgeler dolayısıyla da sonuç büyüklüğü fazla olduğunda arama konusunda Postgresql oldukça yavaş çalışmış olup, Elasticsearch ise oldukça hızlı ve etkin bir performans sergilemiştir.

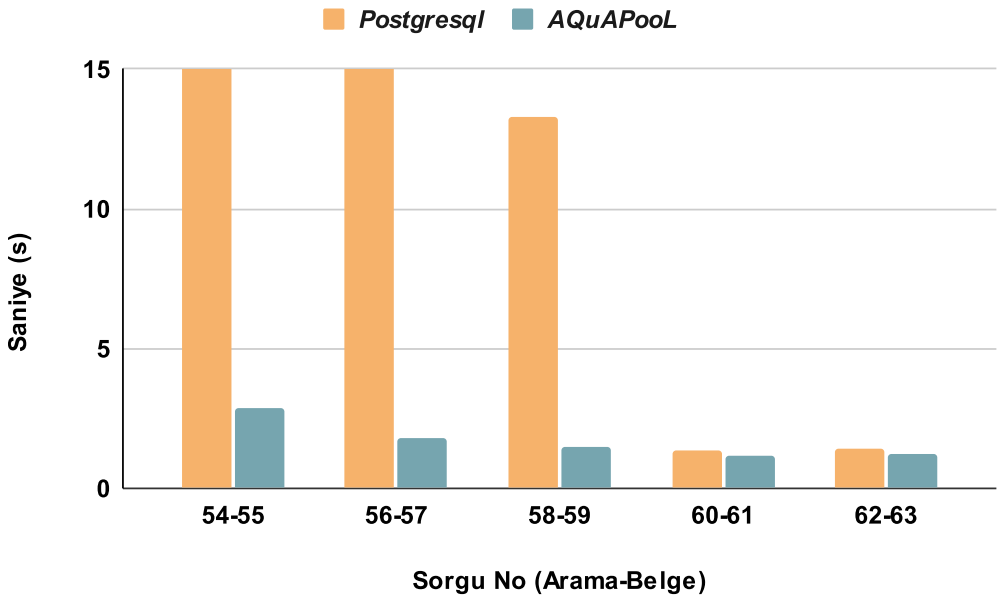
Glomerülonefrit, böbreklerin filtre edici kısmının (glomerulus) iltihaplanması ve hasar görmesidir. Sonuç olarak toksinler, metabolik atıklar ve fazla sıvı idrara uygun şekilde filtrelenmez. Bunun yerine vücutta birikerek şişmeye ve yorgunluğa neden olurlar. Bununla bağlantılı olarak tüm Notevents verisi içinden "Glomerulonephritis" anahtar kelimesi geçen notlar Sorgu-56 ile Postgresql'den, Sorgu-57 ile de Elasticsearch'ten sorgulanmıştır. Sonuçlar incelendiğinde postgresql sorgusu 89.4 saniye sürerken Elasticsearch sorgusu ise 1.82 saniye sürmüştür. Benzer şekilde bu sefer taburcu notları arasından "Glomerulonephritis" anahtar kelimesi geçenler Sorgu-58 ile Postgresql'den, Sorgu-59 ile de Elasticsearch'ten sorgulanmıştır. Bu sorgularda da ilk sorguda olduğu gibi ICD-9 etiketlemesi hedeflenmiştir. Sonuçlara bakıldığında postgresql sorgusu 13.31 saniye, Elasticsearch sorgusu 1.46 saniye sürmüştür. Elasticsearch metin verisi üzerinde indekse sahip olduğundan, sonuç büyüklüğünden bağımsız olarak her zaman etkin, tutarlı ve dengeli sonuçlar vermektedir. Postgresql ilk sorguya benzer olarak Sorgu-56'i de sonuç kümesi büyük olduğundan uzun sürede, Sorgu-58'i ise sonuç kümesi daraldığından daha

hızlı çalıştırmaktadır. Özellikle metin aramalarında arama uzayının daralması Postgresql'in performansını büyük ölçüde etkilemektedir.

Son sorgu tipinde, kan basıncı "blood pressure" ölçütü Postgresql ve Elasticsearch üzerinden önce Sorgu-60 ve Sorgu-61 ile tüm metinler arasından, daha sonra Sorgu-62 ve Sorgu-63 ile taburcu notları arasından sorgulanmıştır. Arama uzayı burada iyice daraltıldığından, Postgresql sırasıyla 1.34 ve 1.41 sn'lik sorgu işletim süresine sahip olup performansı artmış, Elasticsearch ise 1.17 ve 1.22 sn'lik sorgu işletim süresine sahip olarak performansı sabit ve yüksek şekilde kalmıştır.

Sorgu Tanımı	**Postgresql Sorgu No**	**Postgresql İşletimi (s)**	**AQuAPooL Sorgu No**	**AQuAPooL İşletimi (s)**	**Sonuç Boyutu (satır)**
Hypersplenism ile Başlayan NoteEvents Belgeleri	54	101.00	55	2.84	293
Glomerulonephritis ile Başlayan NoteEvents Belgeleri	56	89.40	57	1.82	962
Glomerulonephritis ile Başlayan, *Taburcu Notları* Kategorisindeki NoteEvents Belgeleri	58	13.31	59	1.46	215
Blood Pressure ile Başlayan NoteEvents Belgeleri	60	1.34	61	1.16	3
Blood Pressure ile Başlayan, *Taburcu Notları* Kategorisindeki NoteEvents Belgeleri	62	1.41	63	1.22	1

Tablo 5.4: Postgresql ve AQuAPooL Arama-Belge Tekil Sorgu Sonuçları (s)



Şekil 5.3: Postgresql vs AQuAPooL Arama-Belge Tekil Sorgu Sonucu Kıyaslaması (s)

5.3.2 Çoklu Depo Birleşik Sorgu Sonuçları

Bu kısımda Bölüm-5.2.2.4'de kullanım durumlarından yola çıkılarak farklı depoları barındıran sorgular oluşturulup çoklu depon ortamı üzerinde sınanmıştır. Çoklu deponun asıl etkinliği birleşik sorgular ile ortaya çıkmakta olup sorgu performansı tek depolu yaklaşım ile Postgresql üzerinden kıyaslanmıştır. Oluşturulan çoklu depo ve postgresql sorguları ve sonuçları Tablo-5.5'te listelenmekte olup, kıyaslaması da Şekil-5.4'te görülmektedir.

İlk olarak belirli bir hastanın hastanede yatış yaptığı zamanlarda toplanan magnezyum metrik değeri, etiket değeri ile birlikte Sorgu-64'de Postgresql'den ve Sorgu-65'de ise çoklu depo üzerinden sorgu motoru aracılığıyla sorgulanmıştır. Elde edilen sonuçalra göre Postgresql sorgu sonucu 64.14 saniye, AQuAPooL ile çoklu depo sorgu sonucu ise 5.24 sürmüştür. Dolayısıyla çoklu depo sorgu işletiminin Redis / InfluxDB kombinasyonu ile Postgresql'e göre çok daha yüksek sorgu işletim performansına sahip olduğu görülmektedir. Bu sonuca göre AQuAPooL sorguyu Postgresql'e göre 12 kat daha hızlı çalıştırmıştır.

İkinci sorgu çiftinde sepsis teşhisi konan ve notlarında "azotemia" anahtar kelimesi geçen hastalar Sorgu-66 ile Postgresql'den, Sorgu-67 ile de AQuAPooL ile çoklu depo üzerinden sorgulanmıştır. Azotemi (azot, "nitrojen" + -emi, "kan durumu"), kanda anormal derecede yüksek nitrojen içeren bileşiklerle (üre, kreatinin, çeşitli vücut atığı bileşikleri ve diğer nitrojen açısından zengin bileşikler gibi) karakterize edilen tıbbi bir durumdur. Büyük ölçüde böbrekler tarafından kanın yetersiz veya işlevsiz filtrelenmesi ile alakalıdır. Kontrol altına alınmazsa üremiye (uremia) ve akut böbrek hasarına(acute kidney injury: böbrek yetmezliği) yol açabilir. Böylece kan zehirlenmesi ve azoteminin birlikte görüldüğü hastaların tespit edilmesi amaçlanmıştır. Sorgu sonuçlarına göre Postgresql sorgusu 49 saniye, çoklu depo sorgusu ise 3.64 saniye sürmüştür. Buna göre Redis ve Elasticsearch'ün birlikte sorgulandığı durumda da AQuAPooL sorguyu Postgresql'e göre 13 kat daha hızlı çalıştırmaktadır.

Üçüncü sorgu çiftinde ise bir önceki sorgu genişletilerek sepsis veya şiddetli (severe) sepsis tanısı konan ve notlarında "azotemia" kelimesi geçen hastalar Sorgu-68 ile Postgresql'den, Sorgu-69 ile de AQuAPooL ile çoklu depo üzerinden sorgulanmıştır. Bu sorgu ile azotemi ile sepsis'in birlikte görülme olasılığı olan ve durumu ağır olan hastaların da elde edilmesi ve elde edilen metinlerle ileri çözümlemeler yapılması amaçlanmaktadır. Sonuç kümesi daha da genişlediğinden ötürü Postgresql sorgusu 171 saniyede, AQuAPooL ile Redis ve Elasticsearch'ü içeren çoklu depo sorgusu 14.5 saniyede işletilmiştir. Dolayısıyla çoklu depo üzerinden işletilen sorgu yaklaşık 12 kat daha performanslı çalışmıştır.

Dördüncü sorgu çiftinde de benzer olarak üçüncü sorgu çiftindeki çözümleme gereksinimi genişletilmiş; buna göre sepsis veya şiddetli sepsis tanısı konan, notlarında "azotemia" geçen ve hastanede bakıldığı süre zarfında ölmüş olan hastalar sorgulanmıştır. Azotemi ve sepsis'in birlikte seyrettiği hastalar arasından ölenler listelenerek ölüm oranı tahminlemesi amacıyla çözümleme yapılması da mümkün olmaktadır. Postgresql'in sorgulandığı Sorgu-70 147.4 saniye sürerken, AQuAPooL ile çoklu deponun sorgulandığı Sorgu-71 ise 12.7

saniye sürmüştür. Bir önceki sorgu çiftinin sonuç kümesi içinden ölenler filtrelendiği için sonuç kümesi daraldığından sorgu sonucu beklenen şekilde daha kısa sürmüştür. Redis, Postgresql ve Elasticsearch'ü kapsayan çoklu depo sorgusunun performansının da benzer olarak 11.5 kat daha etkin olduğu gözlemlenmiştir.

Sorgu-72 ve Sorgu-73'da sırasıyla bir sepsis hastasının kabul verisi ile birlikte ona uygulanan tedaviler Postgresql'den ve AQuAPooL ile çoklu depo üzerinden listelenmektedir. Tablo-5.5'deki sonuca göre Postgresql sorgusu 4.5 saniye sürmüşken, AQuAPooL ile Redis ve Postgresql'in sorgulandığı çoklu depo sorgusu ise 3.65 saniye sürmüştür. Sorgu sonuçlarının benzer süreler tutmasının sebebi çoklu depo sorgusunun da bir parçasının ilişkisel veriyi içermesiyle birlikte, Redis ile listelenen tedavilerin sonuç büyüklüğünün az olmasıdır. Dolayısıyla bileşik sorgunun sonuç kümesi küçük olduğundan performans olarak çok büyük bir fark oluşmamıştır. Ancak sonuç kümesi küçük olsa dahi Redis sorgu işletimi için bir performans artışı sağlamıştır.

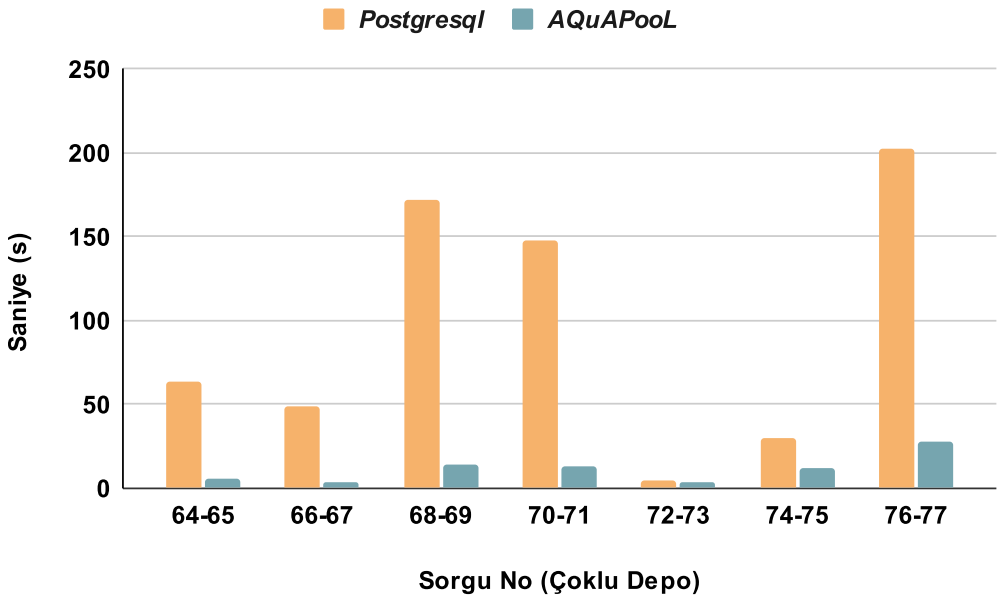
Sorgu-74 ve Sorgu-75'da ise bir önceki sorgu çiftindeki senaryo genişletilerek bir sepsis hastasına uygulanan tedaviler, kabul verisi ve hastanın vücut sıcaklığı değerleri (Temperature F) birlikte Postgresql ve AQuAPooL ile çoklu depo üzerinden sorgulanmaktadır. Sonuç kümesi genişlediğinden ötürü Redis, Postgresql ve InfluxDB depolarını içeren çoklu depo sorgusu 11.8 saniyede işletilmişken Postgresql sorgusu 30.4 saniye sürmüştür. Dolayısıyla çoklu depo 2.5 kat daha yüksek performans sergilemiştir.

Son olarak yedinci sorgu çiftinden tüm depoların sorgulandığı bir senaryo yaratılmıştır. Belirlenen MIMIC-III kullanım durumları ile bağlantılı olarak notlarında "azotemia" geçen hastalardan böbrek yetmezliği olan "Acute kidney failure, unspecified" teşhisi konulan ve hastanede ölmüş olanların hastanede yatmış oldukları belirli tarihler arasındaki solunum hızının (respiratory rate) sorgulanması gerçekleştirilmiştir. Burada azoteminin akut böbrek yaralanmasına, onun da ilerleyerek ölüme sebep olduğu durumların, belirtilerden (semptom) biri olan nefes darlığı ile tespit edilebilmesi için belirli aralıktaki solunum hızı değerleri ölçümleriyle çözümlemesi amaçlanmıştır. Bu kapsamda

Postgresql üzerinde çalıştırılan Sorgu-76 SQL sorgusu 202.3 saniye sürmüşken, Sorgu-77 AQuAPooL çoklu depo sorgusu ise 27.9 saniye sürmüştür ve 3802 satırdan oluşan sonuç kümesi elde edilmiştir. Böylece çoklu deponun gerçek performansı ve etkinliği sonuç büyüklüğünün fazla olduğu bu karmaşık bileşik sorguda 7.25 kat yüksek işletim performansı ile gözler önüne serilmiştir.

Sorgu Tanımı	**Postgresql Sorgu No**	**Postgresql İşletimi (s)**	**AQuAPooL Sorgu No**	**AQuAPooL İşletimi (s)**	**Sonuç Boyutu (satır)**
Bir Hastanın Magnesium Ölçüm Değerleri ve Etiketi	64	64.14	65	5.24	24
Sepsis Teşhisi Konan ve Notlarında "azotemia" Geçen Hastalar	66	49.08	67	3.64	160
Sepsis veya Severe Sepsis Teşhisi Konan ve Notlarında "azotemia" Geçen Hastalar	68	171.70	69	14.52	1230
Sepsis veya Severe Sepsis Teşhisi Konan, Notlarında "azotemia" Geçen ve Hastanede Ölen Hastalar	70	147.45	71	12.74	316
Bir Sepsis Hastasının Kabul Verisi ve Hastaya Uygulanan Tedaviler	72	4.51	73	3.65	13
Bir Sepsis Hastasının Kabul Verisi, Vücut Sıcaklığı Değerleri ve Hastaya Uygulanan Tedaviler	74	30.43	75	11.84	7670
"Acute kidney failure, unspecified" Teşhisi Konan, Notlarında "azotemia" Geçen ve Hastanede Ölen Hastaların Belirli Tarihler Arasındaki Respiratory Rate Değerleri	76	202.31	77	27.90	3802

Tablo 5.5: Postgresql ve AQuAPooL Çoklu Depo Birleşik Sorgu Sonuçları (s)



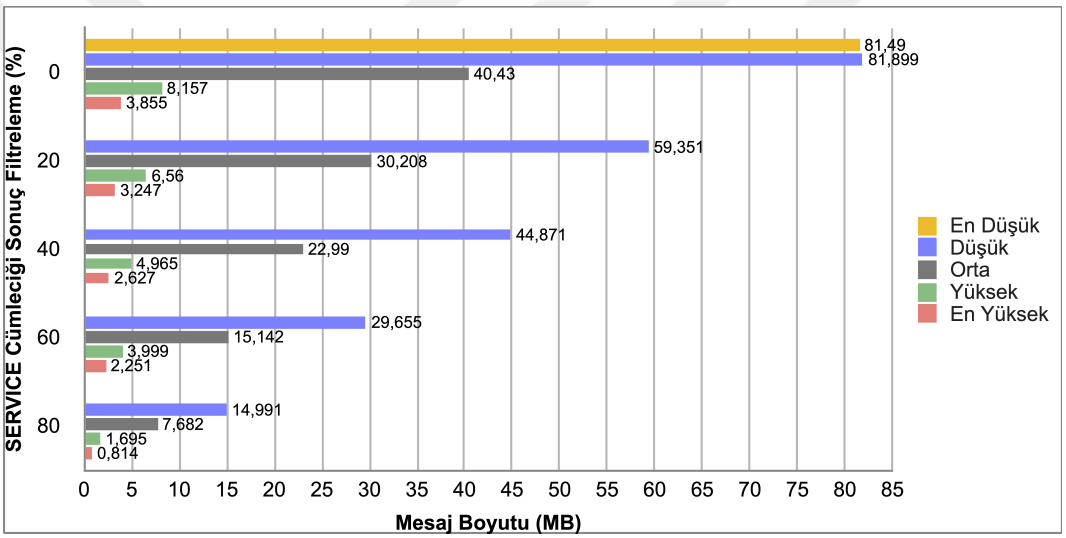
Şekil 5.4: Postgresql vs AQuAPooL Çoklu Depo Birleşik Sorgu Sonucu Kıyaslaması (s)

5.4 Ölçeklenebilirlik Çözümlemesi

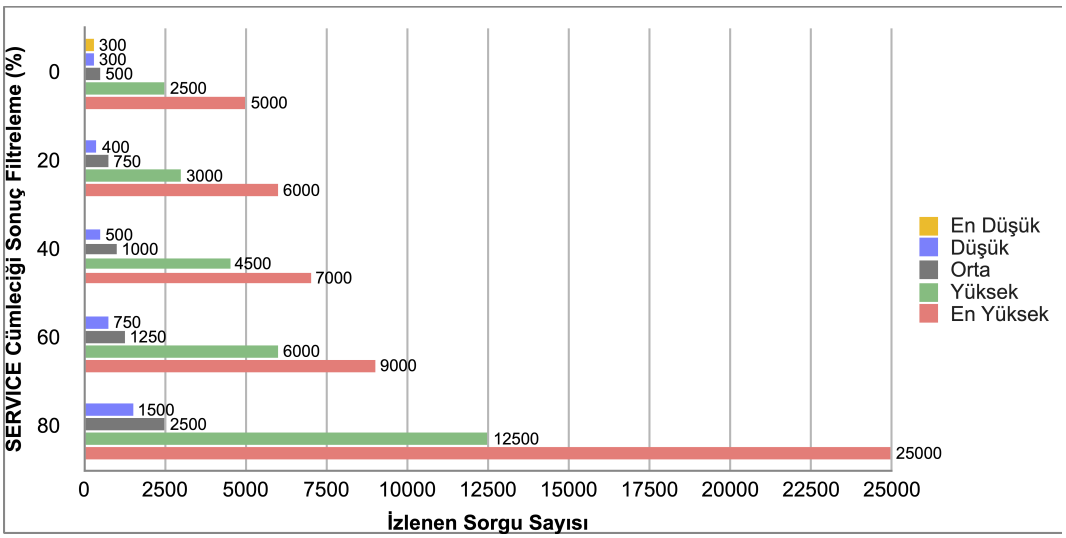
MonARCh aracının mimari detayları ve yapılan yayın çalışması Bölüm-4.3.5'te anlatılmıştı. MonARCh'ın değerlendirme çalışması kapsamında yapılan ölçeklenebilirlik çözümlemesi sonucunda ağ maliyeti sonuçları ile sistemin sınırları ortaya çıkarılmıştır. Buna göre aynı anda işletilebilen sorgu sayısı; ortaya konan bu sonuçlar ve sistem kaynakları ile hesaplanabilmektedir. Tez kapsamında geliştirilen AQuAPooL çoklu depo sisteminde de MonARCh'ın kaynak kodu esas alınarak sorgu takip özelliği devre dışı bırakılmış ve belirlenen depolarla ilgili sorgu işleme ve RDF sonuç dönüşümü eklenerek bağlı veri ile bütünleştirme sağlanmıştır. Her iki araç da aktör model üzerine inşa edilmiş olup Akka küme parçalama yöntemi ile ölçeklenebilirlik yeteneği sağlanmış ve sorgu birleşimi için paralel Grace karma birleşim yöntemi kullanılmıştır. Dolayısıyla iki aracın da ölçeklenebilirlik yeteneği, parametreleri ve sınırları birbirinin aynısıdır. Bu yüzden öncelikle MonARCh'ın ölçeklenebilirlik yeteneği kendi özelinde kurulan borsa kullanım durumu üzerinden açıklanacak olup,

böylece bu sonuçlar ile AQuAPooL'un ölçeklenebilirliği üzerinde çıkarsama yapılabilecektir.

Şekil-5.5 ve Şekil-5.6 ile sisteme gelen farklı seçicilikteki sorgular kullanılarak ve oluşan alt sonuç kümesi üzerinde filtreleme yapıldığında farklı bakış açılarından sistemin ağ çözümlemesi sonuçları görülmektedir. Öncelikle Şekil-5.5 incelendiğinde MonARCh sisteminde sorgu seçiciliği ve onun paralelinde de alt sorgu sonucu filtreleme oranı artınca sorgu başına oluşan toplam mesaj boyutunun azaldığı görülmektedir. Buna paralel olarak Şekil-5.6 incelendiğinde de mesaj boyutu büyüklüğü ile ters orantılı olarak sorgu başına elde edilen sonuç büyüklüğü azaldıkça takip edilen sorgu miktarının arttığı görülmektedir.



Şekil 5.5: Farklı sorgu seçiciliklerine göre ağ üzerinden iletilen sorgu başına toplam mesaj boyutu



Şekil 5.6: Farklı Tipteki Sorgu Seçiciliklerine göre MonARCh'ın Sorgu İzleme Kapasitesi

Bu iki şekildeki ağ analizine göre, sistemin aynı anda işleyebildiği sorgu miktarı bakış açısından ölçeklenebilirlik sınırlarının sorgunun seçiciliği ile doğru orantılı, sorgu başına üretilen mesaj boyutu büyüklüğü ile ise ters orantılı olduğu ortaya konmuştur. Kümedeki her düğümde Intel Xeon E5 2620 v4 (8 çekirdek - 16 iş parçacığı) işlemci, 32 GB RDIMM RAM, RAID 5 yapılandırmalı 1,8 TB 15K SAS Disk ve Ubuntu 18.04 işletim sistemi kuruludur. Aynı zamanda Akka küme parçalama mimarisi de bu iki bilgisayar düğümü üzerine oluşturulmuştur. Bu değerler iki düğümlü küme bilgisayar konfigürasyonu ile elde edilmiştir. MonARCh için ortaya konan bu çözümlemenin, aynı altyapıya ve dolayısıyla da aynı ölçeklenebilirlik parametrelerine sahip olduğundan AQuAPooL için de geçerli olduğu söylenebilecektir. Literatürde ölçeklenebilirlik ile ilgili sınırlar ve olası darboğaz "*veri transferi*" olarak (Karimov et al., 2019) tarafından PolyBench yayınında ortaya konmuştur. Buna göre benzer şekilde tez kapsamında ölçeklenebilirlik ile ilgili yapılan çözümlemelerin ve çıkarsamaların doğru ve tutarlı olduğu sonucuna varılabilmektedir.

5.5 Sistemin Kısıtları

Tez kapsamında ortaya konan AQuAPooL çoklu depo sisteminin anlatılan tüm bu özelliklerinin ve performans ölçümlerinin yanında bazı kısıtları ve sınırları bulunmaktadır. Bunlar kullanılan programlama dili, gerçekleştirim metodolojisi, kütüphaneler, seçilen depoların yetenekleri, kullanılan birleşim algoritması, seçilen çoklu depo sorgu dili vb. olarak sayılabilecektir.

Öncelikle sistemin gerçekleştirim programlama dili olarak Scala kullanılmıştır. Scala programlama dili iş parçacığı güvenli (thread safe) geliştirime olanak sağlar ve aktör model ile programlamaya uygundur. Aktör model gerçekleştirimi olarak da Scala ile yazılmış olan Akka kütüphanesi seçilmiştir. Ancak Scala programlama dili önce Java byte koduna derlendiği için derlemesi daha yavaş olup, Java iş parçacıkları Erlang ve Elixir gibi hafif iş parçacığı ile geliştirim olanağı sunan programlama dillerine göre daha hantaldır. Karşı yönden ise Scala hem nesneye dayalı hem de fonksiyonel bir dildir ve yalnızca fonksiyonel diller olan Erlang ile Elixir'e göre hem geliştirim topluluğu hem de kolaylığı bakımından sayısız üstünlüğü olduğu söylenebilir.

Gerçekleştirim metodolojisi ve kullanılan kütüphaneleri birlikte ele alacak olursak, temelde kullanılan aktör model çok iş parçacığına gereksinim duyan hata durumlarının yoğun olarak görülebildiği mimarilerde etkilidir. Tez kapsamında aktör modelin JVM üzerindeki gerçekleştirimi olan Akka araç seti kullanılmıştır. Akka'nın sayılabilecek en önemli dezavantajı geliştirim metodolojisine ve yapılandırma (configuration) yönetimine bağlı olarak ortaya çıkan öğrenme maliyetidir. Aktör modelin inceliklerini ve ölçeklenebilirlik altyapılarını yönetebilmek için oldukça fazla adam-saat iş gücü harcanmıştır. Aktör modelin üzerine kurulmuş olan Akka'ya alternatif olarak eşle-indirge (map-reduce) mimarisine dayalı Hadoop ve Spark; olay yönelimli (event driven) mimariye dayalı Kafka; gerçek zamanlı işleme (real time processing) mimarilerine dayalı Apache Flink, Storm gibi araçlar bulunmaktadır. Tüm bu araçların ve mimarilerin kendine göre güçlü ve zayıf yönleri ile birlikte öğrenme ve gerçekleştirim maliyetleri mevcuttur. Tezde geliştirilen sistemde

parallel ve eşzamanlı sorgu işleme esas olduğundan aktör modele dayalı Akka en mantıklı seçim olmakla birlikte, alternatif olarak da eşle-indirge mimarisine dayalı Spark ile de gerçekleştirim mümkün görünmektedir. Bunun yanında toplu veri yükleme için Spark kütüphanesi kullanılmış olup, veri tabanlarında paralel sorgu işleyebilmek için her veri deposuna özel bağlantı havuzlama (connection pooling) kütüphaneleri kullanılmıştır.

MIMIC-III verisi için belirlenen kullanım durumlarına göre seçilen depolar ilişkisel veri modeli için Postgresql, anahtar-değer veri modeli için Redis, zaman serisi veri modeli için InfluxDB ve arama-belge veri modeli için Elasticsearch'tür. Seçilen tüm veri depoları kendi veri modelleri bakımından son teknoloji (state of the art) saklama çözümlerindendir. Buradaki tek dezavantaj InfluxDB'nin küme yapılandırması için ticari sürüm gerektirmesidir. Belirlenen tüm bu veri modellerine alternatif saklama çözümleri de tercih edilebilecek olup ilişkisel model için MySQL[56], MariaDB[57], Oracle [58], SQL Server[59]; anahtar değer veri modeli için DynamoDB[60], BoltDB[61], Accumolo[62]; zaman serisi veri modeli için Prometheus[63], TimescaleDB[64], KDB+[65], Druid[66], OpenTSDB[67]; arama-belge veri modeli için OpenSearch[68], Solr[69] gibi çözümler bulunmaktadır. Farklı veri kümesi ve/veya senaryolara göre çoklu depo ortamı yapılandırılacak olursa bu durumdaki gereksinimlere göre farklı saklama çözümleri tercih edilebilecektir. Bunun dışında çoklu depo veri modeli kurgusunun karmaşıklığını azaltmak amacıyla, sistemde

[56]https://www.mysql.com
[57]https://mariadb.org
[58]https://www.oracle.com/database
[59]https://www.microsoft.com/en-us/sql-server
[60]https://aws.amazon.com/dynamodb
[61]https://github.com/boltdb/bolt
[62]https://accumulo.apache.org
[63]https://prometheus.io
[64]https://www.timescale.com
[65]https://kx.com/products/kdb
[66]https://druid.apache.org
[67]http://opentsdb.net
[68]https://opensearch.org
[69]https://solr.apache.org

temel olarak kullanılan saklama çözümüne bir araç takımı (toolkit) veya eklenti (add-on) kurarak bu saklama çözümünün kötü performans gösterdiği veri parçası iyileştirilebilecektir. Örneğin arama gereksinimi için, ilişkisel çözümdeki metin verisi üzerine bir metin arama eklentisi kurarak arama işlevselliği çok daha hızlı gerçekleştirilebilir. Benzer olarak, zaman serisi eklentisi veya uyarlanmış veritabanı (Postgresql üzerine kurulmuş Timescaledb gibi) kullanılarak zaman aralığı sorguları çok daha hızlı işletilebilir. Böylece aktif olarak bir saklama çözümünün (ilişkisel model için Postgresql, MySQL vb.) hâlihazırda kullanıldığı, belirli veri tipleri için dönüşüm işleminin güç olduğu veya kurgulanacak çoklu depo veri modeli karmaşıklığının indirgenmesi gerektiği durumlarda alternatif bir çözüm sağlanmış olacaktır.

Bileşik sorgu işletimi için sistem mimarisinde karma birleşim algoritması kullanılmıştır. Grace algoritması paralelleştirilerek günümüz gereksinimlerine göre güncellenmiş olup aktör mimarisi ile uyumlu çalışacak şekilde uyarlanmıştır. Literatürde iç iç döngülü birleşim, bağlamalı birleşim, sırala-birleştir birleşim gibi kendi içerisinde de farklı ve güncel versiyonları olan algoritmalar bulunmakta olup paralel sorgu işlemeyi desteklediğinden dolayı karma birleşim seçilmiştir. Bu algoritmaların işleyişleri, güçlü ve zayıf yönleri ile birlikte Bölüm-2.5'de ayrıntılı bir şekilde anlatılmıştır ve Bölüm-4.3.2'de tezde kullanılan paralel karma birleşim algoritmasının işleyişi ve neden tercih edildiği detaylıca açıklanmıştır. Alt sorguların seçiciliğinin çok düşük olduğu durumlarda elde edilen ara sonuçlar belleğe sığmayabilir, bu durumda belirli bir kısmının diske yazılması gerekir. Böyle taşma (overflow) durumlarda algoritmanın performansı düşmektedir. Sorgu yeniden yazma olmadığı için tüm alt sorguların seçiciliği sabittir ve takip eden alt sorguların seçiciliğinin artırılarak sonuç kümesinin daraltılma şansı yoktur. Ancak buna rağmen paralel işleme olanağı, sorgu yeniden yazmanın gerekmemesi ve büyük verilerle çalışabilme yeteneğinden ötürü paralel Grace karma birleşim algoritması çoklu depo sisteminde kullanılmaya en uygun yöntemlerden biri olarak öne çıkmaktadır.

Son olarak çoklu depo sorgularını çalıştırmak için ortak bir sorgu dili

bulunmamakta, her deponun kendine özgü sorgu dili kullanılmaktadır. AQuA-PooL sorgu ayrıştırıcı birimine sahip değildir ve sorgu işleme için "alt sorgu - depo" eşlemelerini kullanmaktadır. Bu durum sorgu işleme gerçekleştirimini oldukça basitleştirmekte olup depoların sorgu dilleri dışında farklı bir sorgu dili bilmek gerekmemektedir. Ayrıca doğrudan depoya özgü istenen sorgu eniyileştirilmiş şekilde yazılabildiğinden, dönüşüm işlemindeki performans kayıpları olmamaktadır. Sorgu dönüşümü olmadığından sistemin farklı veri depoları eklenerek genişletilmesi daha kolaydır. Diğer yandan ortak sorgu dili kullanıcılar için sorgulama kolaylığı sağlamaktadır. Ancak sorgu dönüşümü ve yeniden yazma gerektirmektedir ve bu işlevselliklerin etkin yapılabilmesi de tartışmalıdır. Dolayısıyla bu durum sorgulama kolaylığı ile gerçekleştirim kolaylığı ve genişletilebilirlik arasında bir ödünleşmedir (trade-off).

6 SONUÇ VE GELECEK ÇALIŞMALAR

Tez kapsamında çoklu depo sistemleri çözümlenerek sistemin temelindeki çoklu depo ortamı ele alınmıştır. Buna göre çoklu depo ortamınının kurulumu için farklı, karmaşık ve büyük boyutta veri içeren sağlık alanı belirlenmiştir. Sağlık alanındaki önemli veri kümelerinden biri olan MIMIC-III seçilmiş olup çözümlenerek içerdiği veri modelleri ortaya konmuştur. Belirlenen veri modellerine uygun, etkin ve güncel saklama teknolojilerinin dökümantasyonları okunarak püf noktaları ve kullanımları öğrenilmiştir. MIMIC-III veri kümesine referans veren tüm çalışmalar taranarak önemli kullanım durumları belirlenmiştir. Tüm veri kümesi bu kullanım durumlarındaki gereksinimlere göre seçilen depolara yüklenmiştir. Böylece sağlık alanında yoğun bakım birimi kullanım durumu üzerinden bir çoklu depo ortamı oluşturulmuştur.

Çoklu depo sisteminin işlevselliğini ortaya koymak için etkin bir sorgu motoru gerçekleştirimi yapılması hedeflenmiştir. Bu sorgu motoru farklı modeller üzerinde çalışarak depoları sorgulamakta ve buralardan dönen ara sonuçları ortak bir modele (RDF) dönüştürüp birleştirerek ana sonucu oluşturmaktadır. Sorgu motorunun işlevselliğini sergileyebilmek için ilk olarak sorgulama ve birleşim görevlerini doğru şekillerde yerine getirdiği, ayrıca da ölçeklenebilir olarak çalışabildiği gösterilmiştir. Çoklu depo ortamı, doktora süreci içerisinde tamamlanan bir bilimsel araştırma projesi ile birlikte ve sorgu motoru ise doktora sürecinde geliştirilen ve yayın çalışmasına dönüştürülen aktör tabanlı sorgu takip ortamı paralelinde geliştirilmiş ve yürütülmüştür. Yayın çalışmasındaki sonuçlar da ortaya koymaktadır ki aktör tabanlı sorgu takip ortamındaki mimari, çoklu depo ortamı için ölçeklenebilir ve etkin bir sorgulama imkanı sunmuştur. Böylece sorgu takip ortamının temelindeki sorgu motoruna, kurulan ortamın veri modellerine dönüşüm sağlayan sarmalayıcılar dahil edilerek sorgu motoru çoklu depo ortamı üzerinde çalışacak hâle getirilmiştir. Ayrıca geliştirilen çoklu depo sistemi MIMIC-III veri kümesi kullanım durumları çözümlenerek elde edilen sorgularla sınanmıştır. Alınan sonuçlara göre çoklu depo sisteminin etkinliği ve gerekliliği sorgu işleme ve

ölçeklenebilirlik bakış açılarından ortaya konmuştur.

Şekil-4.6'da görüldüğü üzere AQuAPooL'a yollanan her bir ana sorgu küme parçalama mekanizmasını işleten Birleştirici parça bölgesi aracılığıyla küme üzerinde yer alan yalnız bir düğüm üzerinde yönetilir, işletilir ve cevaplanır. Dolayısıyla bir sorgu işleme ardışık düzeninde yer alan Birleştirici, İşletici, Paralel Birleşim Yöneticisi ve Karma Birleşim Uygulayıcısı aktörlerinin tamamı aynı düğüm üzerinde yaratılır, koşulur ve sonlandırılır. Böylece gerçekleştirilecek birleşim işlemi için gerekli sorgu sonuçlarının farklı düğümler üzerinde yer alması durumunda, sonuçların ağ üzerinden iletimi gereksinimi dolayısıyla oluşabilecek maliyetten kaçınılmış olmaktadır.

Bölüm-5.4'te sistemin ölçeklenebilirlik çözümlemesi ortaya konmuştur. Buna göre çoklu depo sistemi geliştirilirken sorgu işleme bakış açısından MonARCh'ın kaynak kodu esas alındığından, MonARCh için yapılan yayın çalışmasında ortaya konan ölçeklenebilirlik sınırları AQuAPooL için de geçerlidir. Dolayısıyla sistemin "*aynı anda işleyebildiği sorgu miktarı*" bakış açısından ölçeklenebilirlik sınırlarının "*sorgunun seçiciliği*" ile doğru orantılı, "*sorgu başına üretilen mesaj boyutu büyüklüğü*" ile ters orantılı olduğu belirlenmiştir.

Son olarak söylemek gerekir ki bir çoklu depo sistemi oluşturabilmek için genel geçer bir yöntem bulunmamaktadır. Alan ve kullanım durumu özelinde çözümleme yapılması gerekmekte olup verilerin sorgulama sıklığı, sorgulanan alanlar, erişim yöntemi ve veriler arasındaki ilişkiler göz önünde bulundurularak gerekli veri modelleri ve saklama çözümleri belirlenebilecektir. Bu durum bir zorluk gibi görünse de karmaşık ve veri yoğun sistemlerin uzun süreli, sağlıklı ve etkin yönetilebilmesi için oldukça kolaylık sağlayacağı düşünüldüğünde, çoklu depo sistemleri en etkili çözümlerden biri olarak karşımıza çıkmaktadır.

Tez kapsamında sunulan AQuAPooL çoklu depo sisteminde kullanım durumuna özel veri dağıtımı, ölçeklenebilir sorgu işletimi, paralel birleşim mekanizması ve kullanım durumuna özel bir değerlendirme çerçevesi geliştirimine odaklanılmıştır. Kullanılan birleşim mekanizması eski ve etkin yöntemlerden olan Grace karma birleşim algoritmasının günümüz donanım şartlarına uyar-

lanmış ve paralel versiyonudur. Bu algoritma sezgisel olarak iyi çalışmakla birlikte literatürde daha güncel ve etkin birleşim algoritmaları mevcuttur. Bu kapsamda gelecekte (Kim et al., 2009) çalışmasındaki paralel radix karma birleşim algoritması ve (Balkesen et al., 2013) kıyaslamada kullandığı m-way sırala-birleştir birleşim algoritması RDF yapısındaki sorgu sonuçlarını birleştirecek şekilde gerçekleştirilecektir. Sonrasında geliştirilen değerlendirme çerçevesinde iki algoritmanın performans kıyaslaması yine (Balkesen et al., 2013) çalışmasındaki metriklerden yararlanılarak yapılacaktır. Buna göre algoritmaların aşağıdaki durumlardaki gibi kıyaslanabilecektir:

- farklı büyüklüklerdeki birleştirilecek ikili sorgu sonuçları için çalışma süreleri
- farklı büyüklüklerdeki birleştirilecek ikili sorgu sonuçları için algoritmaların üretilen sonuç kaydı başına çalışma döngüsü sayısı (cycles per output tuple)
- belirli sorgu sonucu ikilisi işyükü (workload) için farklı iş parçacığı (thread) sayısına göre sonuç üretim verimliliği (saniye başına üretilen sonuç kayıt sayısı) - ***algoritmaların ölçeklenebilirliği***
- artan sorgu sonucu ikilisi büyüklüğüne göre sonuç üretim verimliliği (throughput) - ***algoritmaların ölçeklenebilirliği***

Sonuçlara göre her iki algoritmanın güçlü ve zayıf yanları değerlendirildikten sonra, farklı sonuç yükü ve makine konfigürasyonları gibi durumlara göre daha uygun olan algoritmayı seçip çalıştıran uyarlanabilir (adaptive) bir birleşim mekanizması sorgu işlemcisine dahil edilebilecektir.

Çoklu depo sistemi her ne kadar depoların kendi sorgu dilleri ile ayrı ayrı sorgulanmasını sağlayan bir eşleme veri yapısına sahip olsa da, sistemin SQL benzeri basit bir sorgu sözdiziminin ve bir sorgu ayrıştırıcısının (parser) olması büyük bir kullanım kolaylığı sağlayacaktır. Sisteme yollanacak ana sorgu birden fazla alt sorgudan oluşan birleşik (conjuntive) bir sorgu olabileceğinden, ana sorgunun oluşturulmasında (alt sorgular ve birbirleriyle olan bağlantıları) SQL sorgu dilinin yapılarının kullanılması planlanmaktadır.

Bunun için hazırdaki bir ayrıştırıcı (parser) kütüphanesi (ANTLR[70] vb.) yardımıyla, sistemde kullanılan "sorgu - veri deposu" eşleme veri yapısına benzer olarak çoklu depo sorgusu içinde hem bir depo eşlemesi hem de "birleşim anahtarı" eşlemesi tanımlanacaktır.

Kullanım durumu çözümlemesi sonucu oluşturulan ve değerlendirmede kullanılan sorguların tamamı içsel birleşim (inner join) sorgulardır ve dolayısıyla her iki sonuçta da birleşim anahtarı değeri ile eşleşen kayıtların varlığına bakılmaktadır. Bunun yanında sol birleşim ile ilk, sağ birleşim ile ikinci ve tam birleşim ile her iki sonuç üzerinde eşleşmeyen birleşim anahtarı değerleri de sonuca dahil edilir. Sistemin sorgu sözdizimi tanımlandıktan sonra; sol birleşim (left join), sağ birleşim (right join) ve tam birleşim (full join) tipinde sorgular ile sistemin değerlendirmesinin genişletilmesi planlanmaktadır.

Çoklu depo sistemine gelen sorguların daha etkin çalıştırabilmesi için bir sorgu eniyileştirme yaklaşımının olması büyük bir avantaj olacaktır. Dolayısıyla her bir sorgu deposunun sistemdeki işletici yapılarına özel birer sorgu eniyileştirici mekanizması eklenerek depolar üzerinde işletilecek sorguların daha kısa sürede tamamlanması hedeflenmektedir.

Bu mimari iyileştirmelerin yanında değerlendirme aşaması genişletilerek, MonARCh'ta yapılana benzer olarak AQuAPooL sisteminin ölçeklenebilirlik yeteneği farklı seçicilikteki sorgular yoğun bir şekilde sisteme yollanarak ölçülecektir. Böylece aynı anda işletilebilen sorgu miktarı geliştirilen değerlendirme çerçevesindeki sorgular cinsinden ortaya konacaktır.

Son olarak şimdiye kadar yapılan geliştirimler kapsamında sistemin mimarisinin, yeteneklerinin, özgün yanlarının anlatıldığı ve yapılan değerlendirme kapsamında alınan sonuçlara göre etkinliğinin ispatlanarak açıklandığı bir dergi yayını çalışması ortaya konacaktır. Bu çalışmaya ek olarak ise; MIMIC-III veri kümesine referans veren çalışmaların çözümlenmesi sonucunda ortaya konan kullanım durumlarının detaylı olarak anlatılıp açıklandığı ve bu kullanım durumlarına göre önemli çalışmaları sınıflandıran bir araştırma/inceleme (survey) yayını çalışması yapılması planlanmaktadır.

[70] https://www.antlr.org

KAYNAKLAR DİZİNİ

Agrawal, D., **Chawla, S.**, **Contreras-Rojas, B.**, **Elmagarmid, A.**, **Idris, Y.**, **Kaoudi, Z.**, **Kruse, S.**, **Lucas, J.**, **Mansour, E.**, **Ouzzani, M.**, **et al.**, 2018, RHEEM: enabling cross-platform data processing: may the big data be with you!, *Proceedings of the VLDB Endowment*, 11(11), 1414–1427.

Agrawal, M., **Hegselmann, S.**, **Lang, H.**, **Kim, Y.**, and **Sontag, D.**, 2022, Large language models are few-shot clinical information extractors, in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1998–2022.

Akar, Z., **Halaç, T.G.**, **Ekinci, E.E.**, and **Dikenelli, O.**, 2012, Querying the Web of Interlinked Datasets using VOID Descriptions., *LDOW*, 937.

Alotaibi, R., **Bursztyn, D.**, **Deutsch, A.**, **Manolescu, I.**, and **Zampetakis, S.**, 2019, Towards scalable hybrid stores: constraint-based rewriting to the rescue, in *Proceedings of the 2019 International Conference on Management of Data*, pages 1660–1677.

Alotaibi, R., **Cautis, B.**, **Deutsch, A.**, **Latrache, M.**, **Manolescu, I.**, and **Yang, Y.**, 2020, ESTOCADA: towards scalable polystore systems, *Proceedings of the VLDB Endowment*, 13(12), 2949–2952.

Alsentzer, E., **Murphy, J.R.**, **Boag, W.**, **Weng, W.H.**, **Jin, D.**, **Naumann, T.**, and **McDermott, M.**, 2019, Publicly available clinical BERT embeddings, *arXiv preprint arXiv:1904.03323*.

Arenas, M. and **Pérez, J.**, 2012, *Federation and Navigation in SPARQL 1.1*, Springer.

Auer, S., **Bizer, C.**, **Kobilarov, G.**, **Lehmann, J.**, **Cyganiak, R.**, and **Ives, Z.**, 2007, Dbpedia: A nucleus for a web of open data, in *international semantic web conference*, pages 722–735, Springer.

KAYNAKLAR DİZİNİ (devam)

Balkesen, C., **Alonso, G.**, **Teubner, J.**, and **Özsu, M.T.**, 2013, Multi-core, main-memory joins: Sort vs. hash revisited, *Proceedings of the VLDB Endowment*, 7(1), 85–96.

Basciani, F., **Di Rocco, J.**, **Di Ruscio, D.**, **Pierantonio, A.**, and **Iovino, L.**, 2020, TyphonML: a modeling environment to develop hybrid polystores, in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 1–5.

Baumel, T., **Nassour-Kassis, J.**, **Cohen, R.**, **Elhadad, M.**, and **Elhadad, N.**, 2017, Multi-label classification of patient notes a case study on ICD code assignment, *arXiv preprint arXiv:1709.09587*.

Beaulieu-Jones, B.K., **Wu, Z.S.**, **Williams, C.**, **Lee, R.**, **Bhavnani, S.P.**, **Byrd, J.B.**, and **Greene, C.S.**, 2019, Privacy-preserving generative deep neural networks support clinical data sharing, *Circulation: Cardiovascular Quality and Outcomes*, 12(7), e005122.

Begoli, E., **Camacho-Rodríguez, J.**, **Hyde, J.**, **Mior, M.J.**, and **Lemire, D.**, 2018, Apache calcite: A foundational framework for optimized query processing over heterogeneous data sources, in *Proceedings of the 2018 International Conference on Management of Data*, pages 221–230.

Ben Hamadou, H., **Gallinucci, E.**, and **Golfarelli, M.**, 2019, Answering GPSJ queries in a polystore: A dataspace-based approach, in *International Conference on Conceptual Modeling*, pages 189–203, Springer.

Benats, P., **Meurice, L.**, and **Cleve, A.**, 2022, Query-based Schema Evolution Recommendations for Hybrid Polystores, in *Proceedings of the 41st International Conference on Conceptual Modeling (ER 2022), Forum Track*.

KAYNAKLAR DİZİNİ (devam)

Berman, J.J., 2013, *Principles of big data: preparing, sharing, and analyzing complex information*, Newnes.

Berners-Lee, T., 2006, Linked data-design issues, *http://www.w3.org/DesignIssues/LinkedData. html*.

Bizer, C., **Heath, T.**, and **Berners-Lee, T.**, 2011, Linked data: The story so far, in *Semantic services, interoperability and web applications: emerging concepts*, pages 205–227, IGI global.

Bizer, C., **Lehmann, J.**, **Kobilarov, G.**, **Auer, S.**, **Becker, C.**, **Cyganiak, R.**, and **Hellmann, S.**, 2009, Dbpedia-a crystallization point for the web of data, *Journal of web semantics*, 7(3), 154–165.

Bonaque, R., **Cao, T.D.**, **Cautis, B.**, **Goasdoué, F.**, **Letelier, J.**, **Manolescu, I.**, **Mendoza, O.**, **Ribeiro, S.**, **Tannier, X.**, and **Thomazo, M.**, 2016, Mixed-instance querying: a lightweight integration architecture for data journalism, in *VLDB*.

Bondiombouy, C., **Kolev, B.**, **Levchenko, O.**, and **Valduriez, P.**, 2015, Integrating big data and relational data with a functional sql-like query language, in *International Conference on Data Management in Cloud, Grid and P2P Systems*, pages 170–185, Springer.

Bondiombouy, C., **Kolev, B.**, **Levchenko, O.**, and **Valduriez, P.**, 2016, Multistore big data integration with CloudMdsQL, *Transactions on Large-Scale Data-and Knowledge-Centered Systems XXVIII: Special Issue on Database-and Expert-Systems Applications*, pages 48–74.

Bondiombouy, C. and **Valduriez, P.**, 2016, Query processing in multistore systems: an overview, *International Journal of Cloud Computing*, 5(4), 309–346.

KAYNAKLAR DİZİNİ (devam)

Bugiotti, F., **Bursztyn, D.**, **Deutsch, A.**, **Ileana, I.**, and **Manolescu, I.**, 2015a, Invisible glue: scalable self-tuning multi-stores, in *Conference on innovative data systems research (CIDR)*.

Bugiotti, F., **Bursztyn, D.**, **Deutsch, A.**, **Ileana, I.**, and **Manolescu, I.**, 2015b, Toward Scalable Hybrid Stores, in *SEBD Italian Symposium on Advanced Database Systems*.

Bugiotti, F., **Bursztyn, D.**, **Deutsch, A.**, **Manolescu, I.**, and **Zampetakis, S.**, 2016, Flexible hybrid stores: Constraint-based rewriting to the rescue, in *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 1394–1397, IEEE.

Cafarella, M., **DeWitt, D.**, **Gadepally, V.**, **Kepner, J.**, **Kozyrakis, C.**, **Kraska, T.**, **Stonebraker, M.**, and **Zaharia, M.**, 2021, A polystore based database operating system (DBOS), in *Heterogeneous Data Management, Polystores, and Analytics for Healthcare: VLDB Workshops, Poly 2020 and DMAH 2020, Virtual Event, August 31 and September 4, 2020, Revised Selected Papers 6*, pages 3–24, Springer.

Che, Z., **Purushotham, S.**, **Cho, K.**, **Sontag, D.**, and **Liu, Y.**, 2018, Recurrent neural networks for multivariate time series with missing values, *Scientific reports*, 8(1), 6085.

Chen, P., **Gadepally, V.**, and **Stonebraker, M.**, 2016, The bigdawg monitoring framework, in *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, IEEE.

Chu, S., **Balazinska, M.**, and **Suciu, D.**, 2015, From theory to practice: Efficient join query evaluation in a parallel database system, in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 63–78.

KAYNAKLAR DİZİNİ (devam)

Dasgupta, S., **Coakley, K.**, and **Gupta, A.**, 2016, Analytics-driven data ingestion and derivation in the AWESOME polystore, in *2016 IEEE International Conference on Big Data (Big Data)*, pages 2555–2564, IEEE.

Desautels, T., **Calvert, J.**, **Hoffman, J.**, **Jay, M.**, **Kerem, Y.**, **Shieh, L.**, **Shimabukuro, D.**, **Chettipally, U.**, **Feldman, M.D.**, **Barton, C.**, **et al.**, 2016, Prediction of sepsis in the intensive care unit with minimal electronic health record data: a machine learning approach, *JMIR medical informatics*, 4(3), e5909.

Du, J., **Meehan, J.**, **Tatbul, N.**, and **Zdonik, S.**, 2017, Towards dynamic data placement for polystore ingestion, in *Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics*, pages 1–8.

Duggan, J., **Elmore, A.J.**, **Stonebraker, M.**, **Balazinska, M.**, **Howe, B.**, **Kepner, J.**, **Madden, S.**, **Maier, D.**, **Mattson, T.**, and **Zdonik, S.**, 2015, The bigdawg polystore system, *ACM Sigmod Record*, 44(2), 11–16.

Dziedzic, A., **Duggan, J.**, **Elmore, A.J.**, **Gadepally, V.**, and **Stonebraker, M.**, 2015, Bigdawg: a polystore for diverse interactive applications, in *Data Syst. Interactive Anal. Workshop*, volume 2015.

Dziedzic, A., **Elmore, A.J.**, and **Stonebraker, M.**, 2016, Data transformation and migration in polystores, in *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, IEEE.

Fink, J., **Gobert, M.**, and **Cleve, A.**, 2020, Adapting queries to database schema changes in hybrid polystores, in *2020 IEEE 20th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 127–131, IEEE.

Franklin, M., **Halevy, A.**, and **Maier, D.**, 2005, From databases to

KAYNAKLAR DİZİNİ (devam)

dataspaces: a new abstraction for information management, *ACM Sigmod Record*, 34(4), 27–33.

Gadepally, V., **Chen, P.**, **Duggan, J.**, **Elmore, A.**, **Haynes, B.**, **Kepner, J.**, **Madden, S.**, **Mattson, T.**, and **Stonebraker, M.**, 2016, The BigDAWG polystore system and architecture, in *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, IEEE.

Gadepally, V., **Kepner, J.**, **Arcand, W.**, **Bestor, D.**, **Bergeron, B.**, **Byun, C.**, **Edwards, L.**, **Hubbell, M.**, **Michaleas, P.**, **Mullen, J.**, **et al.**, 2015, D4M: Bringing associative arrays to database engines, in *2015 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, IEEE.

Gadepally, V., **O'Brien, K.**, **Dziedzic, A.**, **Elmore, A.**, **Kepner, J.**, **Madden, S.**, **Mattson, T.**, **Rogers, J.**, **She, Z.**, and **Stonebraker, M.**, 2017, BigDAWG version 0.1, in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, IEEE.

Goldberger, A.L., **Amaral, L.A.**, **Glass, L.**, **Hausdorff, J.M.**, **Ivanov, P.C.**, **Mark, R.G.**, **Mietus, J.E.**, **Moody, G.B.**, **Peng, C.K.**, and **Stanley, H.E.**, 2000, PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals, *circulation*, 101(23), e215–e220.

Gubanov, M., 2017, Polyfuse: A large-scale hybrid data fusion system, in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 1575–1578, IEEE.

Gupta, A.M., **Gadepally, V.**, and **Stonebraker, M.**, 2016, Cross-engine query execution in federated database systems, in *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, IEEE.

KAYNAKLAR DİZİNİ (devam)

Gupta, M., 2012, *Akka essentials*, Packt Publishing Ltd.

Haas, L.M., **Kossmann, D.**, **Wimmers, E.L.**, and **Yang, J.**, 1997, Optimizing queries across diverse data sources, in *VLDB*, volume 97, pages 25–29.

Halperin, D., **Teixeira de Almeida, V.**, **Choo, L.L.**, **Chu, S.**, **Koutris, P.**, **Moritz, D.**, **Ortiz, J.**, **Ruamviboonsuk, V.**, **Wang, J.**, **Whitaker, A.**, **et al.**, 2014, Demonstration of the Myria big data management service, in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 881–884.

Harutyunyan, H., **Khachatrian, H.**, **Kale, D.C.**, **Ver Steeg, G.**, and **Galstyan, A.**, 2019, Multitask learning and benchmarking with clinical time series data, *Scientific data*, 6(1), 96.

Hausenblas, M. and **Nadeau, J.**, 2013, Apache drill: interactive ad-hoc analysis at scale, *Big data*, 1(2), 100–104.

Helland, P., 2007, Life beyond Distributed Transactions: an Apostate's Opinion., in *CIDR*, volume 2007, pages 132–141.

Helland, P., 2016, Life beyond distributed transactions: an apostate's opinion, *Queue*, 14(5), 69–98.

Hewitt, C., 2010, Actor model of computation: scalable robust information systems, *arXiv preprint arXiv:1008.1459*.

Hewitt, C., **Bishop, P.**, and **Steiger, R.**, 1973, A universal modular actor formalism for artificial intelligence, in *Proceedings of the 3rd international joint conference on Artificial intelligence*, pages 235–245.

Huang, S.C., **Shen, L.**, **Lungren, M.P.**, and **Yeung, S.**, 2021, Gloria: A multimodal global-local representation learning framework for label-efficient

KAYNAKLAR DİZİNİ (devam)

medical image recognition, in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3942–3951.

Idreos, S., **Maas, L.M.**, and **Kester, M.S.**, 2017, Evolutionary data systems, *arXiv preprint arXiv:1706.05714*.

İnan, E., **Yönyül, B.**, and **Tekbacak, F.**, 2017, Çoklu Veri Depo Ortamını Kullanan Alana Özgü Varlık Bağlama Yaklaşımı, in *8. Akıllı Sistemlerde Yenilikler ve Uygulamaları Konferansı (ASYU-2017)*.

İnan, E., **Yönyül, B.**, and **Tekbacak, F.**, 2018, A Domain Specific Entity Linking Approach Consuming Multistore Environment, *Akıllı Sistemler ve Uygulamaları Dergisi (Journal of Intelligent Systems with Applications)*, 1(1), 46–52.

Johnson, A., **Bulgarelli, L.**, **Pollard, T.**, **Horng, S.**, **Celi, L.A.**, and **Mark, R.**, 2020, Mimic-iv, *PhysioNet. Available online at: https://physionet. org/content/mimiciv/1.0/(accessed August 23, 2021)*.

Johnson, A.E., **Bulgarelli, L.**, **Shen, L.**, **Gayles, A.**, **Shammout, A.**, **Horng, S.**, **Pollard, T.J.**, **Hao, S.**, **Moody, B.**, **Gow, B.**, **et al.**, 2023, MIMIC-IV, a freely accessible electronic health record dataset, *Scientific data*, 10(1), 1.

Johnson, A.E., **Pollard, T.J.**, and **Mark, R.G.**, 2017, Reproducibility in critical care: a mortality prediction case study, in *Machine learning for healthcare conference*, pages 361–376, PMLR.

Johnson, A.E., **Pollard, T.J.**, **Shen, L.**, **Lehman, L.w.H.**, **Feng, M.**, **Ghassemi, M.**, **Moody, B.**, **Szolovits, P.**, **Anthony Celi, L.**, and **Mark, R.G.**, 2016, MIMIC-III, a freely accessible critical care database, *Scientific data*, 3(1), 1–9.

KAYNAKLAR DİZİNİ (devam)

Kaitoua, A., **Rabl, T.**, **Katsifodimos, A.**, and **Markl, V.**, 2019, Muses: distributed data migration system for polystores, in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1602–1605, IEEE.

Karimov, J., **Rabl, T.**, and **Markl, V.**, 2019, Polybench: The first benchmark for polystores, in *Performance Evaluation and Benchmarking for the Era of Artificial Intelligence: 10th TPC Technology Conference, TPCTC 2018, Rio de Janeiro, Brazil, August 27–31, 2018, Revised Selected Papers 10*, pages 24–41, Springer.

Kepner, J., **Gadepally, V.**, **Hutchison, D.**, **Jananthan, H.**, **Mattson, T.**, **Samsi, S.**, and **Reuther, A.**, 2016, Associative array model of SQL, NoSQL, and NewSQL Databases, in *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–9, IEEE.

Khan, Y., **Zimmermann, A.**, **Jha, A.**, **Gadepally, V.**, **d'Aquin, M.**, and **Sahay, R.**, 2019, One size does not fit all: querying web polystores, *Ieee Access*, 7, 9598–9617.

Khurshid, S., **Reeder, C.**, **Harrington, L.X.**, **Singh, P.**, **Sarma, G.**, **Friedman, S.F.**, **Di Achille, P.**, **Diamant, N.**, **Cunningham, J.W.**, **Turner, A.C.**, **et al.**, 2022, Cohort design and natural language processing to reduce bias in electronic health records research, *NPJ Digital Medicine*, 5(1), 47.

Kim, C., **Kaldewey, T.**, **Lee, V.W.**, **Sedlar, E.**, **Nguyen, A.D.**, **Satish, N.**, **Chhugani, J.**, **Di Blas, A.**, and **Dubey, P.**, 2009, Sort vs. hash revisited: Fast join implementation on modern multi-core CPUs, *Proceedings of the VLDB Endowment*, 2(2), 1378–1389.

Kitsuregawa, M., **Tanaka, H.**, and **Moto-Oka, T.**, 1983, Application of

KAYNAKLAR DİZİNİ (devam)

hash to data base machine and its architecture, *New Generation Computing*, 1, 63–74.

Kolev, B., **Bondiombouy, C.**, **Levchenko, O.**, **Valduriez, P.**, **Jimenez-Péris, R.**, **Pau, R.**, and **Pereira, J.**, 2016a, Design and implementation of the CloudMdsQL multistore system, in *CLOSER: Cloud Computing and Services Science*, volume 1, pages 352–359.

Kolev, B., **Bondiombouy, C.**, **Valduriez, P.**, **Jiménez-Peris, R.**, **Pau, R.**, and **Pereira, J.**, 2016b, The cloudmdsql multistore system, in *Proceedings of the 2016 International Conference on Management of Data*, pages 2113–2116.

Kolev, B., **Levchenko, O.**, **Pacitti, E.**, **Valduriez, P.**, **Vilaça, R.**, **Gonçalves, R.**, **Jiménez-Peris, R.**, and **Kranas, P.**, 2018, Parallel polyglot query processing on heterogeneous cloud data stores with LeanXcale, in *2018 IEEE International Conference on Big Data (Big Data)*, pages 1757–1766, IEEE.

Kolev, B., **Pau, R.**, **Levchenko, O.**, **Valduriez, P.**, **Jiménez-Peris, R.**, and **Pereira, J.**, 2016c, Benchmarking polystores: the CloudMdsQL experience, in *2016 IEEE International Conference on Big Data (Big Data)*, pages 2574–2579, IEEE.

Kolev, B., **Valduriez, P.**, **Bondiombouy, C.**, **Jiménez-Peris, R.**, **Pau, R.**, and **Pereira, J.**, 2016d, CloudMdsQL: querying heterogeneous cloud data stores with a common language, *Distributed and parallel databases*, 34, 463–503.

Kolovos, D., **Medhat, F.**, **Paige, R.**, **Di Ruscio, D.**, **Van Der Storm, T.**, **Scholze, S.**, and **Zolotas, A.**, 2019, Domain-specific languages for the design, deployment and manipulation of heterogeneous databases, in

KAYNAKLAR DİZİNİ (devam)

2019 IEEE/ACM 11th International Workshop on Modelling in Software Engineering (MiSE), pages 89–92, IEEE.

Komorowski, M., **Celi, L.A.**, **Badawi, O.**, **Gordon, A.C.**, and **Faisal, A.A.**, 2018, The artificial intelligence clinician learns optimal treatment strategies for sepsis in intensive care, *Nature medicine*, 24(11), 1716–1720.

Kong, G., **Lin, K.**, and **Hu, Y.**, 2020, Using machine learning methods to predict in-hospital mortality of sepsis patients in the ICU, *BMC medical informatics and decision making*, 20, 1–10.

Kranas, P., **Kolev, B.**, **Levchenko, O.**, **Pacitti, E.**, **Valduriez, P.**, **Jiménez-Peris, R.**, and **Patiño-Martinez, M.**, 2021, Parallel query processing in a polystore, *Distributed and Parallel Databases*, pages 1–39.

Leclercq, É. and **Savonnet, M.**, 2018, A tensor based data model for polystore: an application to social networks data, in *Proceedings of the 22nd International Database Engineering & Applications Symposium*, pages 110–118.

Leclercq, E. and **Savonnet, M.**, 2019, TDM: a tensor data model for logical data independence in polystore systems, in *Heterogeneous Data Management, Polystores, and Analytics for Healthcare: VLDB 2018 Workshops, Poly and DMAH, Rio de Janeiro, Brazil, August 31, 2018, Revised Selected Papers 4*, pages 39–56, Springer.

Lehmann, J., **Isele, R.**, **Jakob, M.**, **Jentzsch, A.**, **Kontokostas, D.**, **Mendes, P.N.**, **Hellmann, S.**, **Morsey, M.**, **Van Kleef, P.**, **Auer, S.**, **et al.**, 2015, Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia, *Semantic web*, 6(2), 167–195.

Lenzerini, M., 2002, Data integration: A theoretical perspective, in *Proce-*

KAYNAKLAR DİZİNİ (devam)

edings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pages 233–246.

Lu, Q., **De Silva, N.**, **Kafle, S.**, **Cao, J.**, **Dou, D.**, **Nguyen, T.H.**, **Sen, P.**, **Hailpern, B.**, **Reinwald, B.**, and **Li, Y.**, 2019, Learning electronic health records through hyperbolic embedding of medical ontologies, in *Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, pages 338–346.

Luo, Y., 2017, Recurrent neural networks for classifying relations in clinical notes, *Journal of biomedical informatics*, 72, 85–95.

Luo, Y., 2022, Evaluating the state of the art in missing data imputation for clinical data, *Briefings in Bioinformatics*, 23(1), bbab489.

Maccioni, A., **Basili, E.**, and **Torlone, R.**, 2016, QUEPA: QUerying and exploring a polystore by augmentation, in *Proceedings of the 2016 International Conference on Management of Data*, pages 2133–2136.

Maccioni, A. and **Torlone, R.**, 2018, Augmented access for querying and exploring a polystore, in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 77–88, IEEE.

Manyika, J., **Chui, M.**, **Brown, B.**, **Bughin, J.**, **Dobbs, R.**, **Roxburgh, C.**, **Hung Byers, A.**, **et al.**, 2011, *Big data: The next frontier for innovation, competition, and productivity*, McKinsey Global Institute.

Melnik, S., **Gubarev, A.**, **Long, J.J.**, **Romer, G.**, **Shivakumar, S.**, **Tolton, M.**, and **Vassilakis, T.**, 2010, Dremel: interactive analysis of web-scale datasets, *Proceedings of the VLDB Endowment*, 3(1-2), 330–339.

Mullenbach, J., **Wiegreffe, S.**, **Duke, J.**, **Sun, J.**, and **Eisenstein, J.**, 2018, Explainable prediction of medical codes from clinical text, *arXiv preprint arXiv:1802.05695*.

KAYNAKLAR DİZİNİ (devam)

Nemati, S., **Holder, A.**, **Razmi, F.**, **Stanley, M.D.**, **Clifford, G.D.**, and **Buchman, T.G.**, 2018, An interpretable machine learning model for accurate prediction of sepsis in the ICU, *Critical care medicine*, 46(4), 547.

Ong, K.W., **Papakonstantinou, Y.**, and **Vernoux, R.**, 2014a, The SQL++ query language: Configurable, unifying and semi-structured, *arXiv preprint arXiv:1405.3631*.

Ong, K.W., **Papakonstantinou, Y.**, and **Vernoux, R.**, 2014b, The SQL++ Semi-structured Data Model and Query Language: A Capabilities Survey of SQL-on-Hadoop, NoSQL and NewSQL Databases.

Ong, K.W., **Papakonstantinou, Y.**, and **Vernoux, R.**, 2014c, The SQL++ unifying semi-structured query language, and an expressiveness benchmark of SQL-on-Hadoop, NoSQL and NewSQL databases, *CoRR, abs/1405.3631*.

Papakonstantinou, Y., 2016, Polystore Query Rewriting: The Challenges of Variety., in *EDBT/ICDT Workshops*, Citeseer.

Podkorytov, M., **Soderman, D.**, and **Gubanov, M.**, 2017, Hybrid. poly: An interactive large-scale in-memory analytical polystore, in *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 43–50, IEEE.

Poudel, M., **Sarode, R.P.**, **Shrestha, S.**, **Chu, W.**, and **Bhalla, S.**, 2019, Development of a polystore data management system for an evolving big scientific data archive, in *Heterogeneous Data Management, Polystores, and Analytics for Healthcare: VLDB 2019 Workshops, Poly and DMAH, Los Angeles, CA, USA, August 30, 2019, Revised Selected Papers 5*, pages 167–182, Springer.

Poudel, M., **Sarode, R.P.**, **Watanobe, Y.**, **Mozgovoy, M.**, and **Bhalla,**

KAYNAKLAR DİZİNİ (devam)

S., 2022, Processing analytical queries over polystore system for a large astronomy data repository, *Applied Sciences*, 12(5), 2663.

Prakash, A., **Zhao, S.**, **Hasan, S.**, **Datla, V.**, **Lee, K.**, **Qadir, A.**, **Liu, J.**, and **Farri, O.**, 2017, Condensed memory networks for clinical diagnostic inferencing, in *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

Prasad, N., **Cheng, L.F.**, **Chivers, C.**, **Draugelis, M.**, and **Engelhardt, B.E.**, 2017, A reinforcement learning approach to weaning of mechanical ventilation in intensive care units, *arXiv preprint arXiv:1704.06300*.

Prud'hommeaux, E., **Seaborne, A.**, **et al.**, 2017, SPARQL query language for RDF. W3C Recommendation (2008).

Raghu, A., **Komorowski, M.**, **Celi, L.A.**, **Szolovits, P.**, and **Ghassemi, M.**, 2017, Continuous state-space models for optimal sepsis treatment: a deep reinforcement learning approach, in *Machine Learning for Healthcare Conference*, pages 147–163, PMLR.

Rajkomar, A., **Oren, E.**, **Chen, K.**, **Dai, A.M.**, **Hajaj, N.**, **Hardt, M.**, **Liu, P.J.**, **Liu, X.**, **Marcus, J.**, **Sun, M.**, **et al.**, 2018, Scalable and accurate deep learning with electronic health records, *NPJ digital medicine*, 1(1), 18.

Ricci, A., **Piunti, M.**, **Viroli, M.**, and **Omicini, A.**, 2009, Environment programming in CArtAgO, *Multi-agent programming: Languages, tools and applications*, pages 259–288.

Sadalage, P.J. and **Fowler, M.**, 2013, *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*, Pearson Education.

Saeed, M., **Villarroel, M.**, **Reisner, A.T.**, **Clifford, G.**, **Lehman, L.W.**, **Moody, G.**, **Heldt, T.**, **Kyaw, T.H.**, **Moody, B.**, and **Mark, R.G.**,

KAYNAKLAR DİZİNİ (devam)

2011, Multiparameter Intelligent Monitoring in Intensive Care II (MIMIC-II): a public-access intensive care unit database, *Critical care medicine*, 39(5), 952.

Schmidt, M., **Görlitz, O.**, **Haase, P.**, **Ladwig, G.**, **Schwarte, A.**, and **Tran, T.**, 2011, Fedbench: A benchmark suite for federated semantic data query processing, in *The Semantic Web–ISWC 2011: 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I 10*, pages 585–600, Springer.

She, Z., **Ravishankar, S.**, and **Duggan, J.**, 2016, BigDAWG polystore query optimization through semantic equivalences, in *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, IEEE.

Singhal, R., **Zhang, N.**, **Nardi, L.**, **Shahbaz, M.**, and **Olukotun, K.**, 2019, Polystore++: Accelerated polystore system for heterogeneous workloads, in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1641–1651, IEEE.

Tan, R., **Chirkova, R.**, **Gadepally, V.**, and **Mattson, T.G.**, 2017, Enabling query processing across heterogeneous data models: A survey, in *2017 IEEE International Conference on Big Data (Big Data)*, pages 3211–3220, IEEE.

Tomašev, N., **Harris, N.**, **Baur, S.**, **Mottram, A.**, **Glorot, X.**, **Rae, J.W.**, **Zielinski, M.**, **Askham, H.**, **Saraiva, A.**, **Magliulo, V.**, **et al.**, 2021, Use of deep learning to develop continuous-risk models for adverse event prediction from electronic health records, *Nature Protocols*, 16(6), 2765–2787.

Ünalir, M.O., **İnan, E.**, **Yönyül, B.**, **Olca, E.**, **Sentürk, F.**, **Mostafapour, V.**, **Yildiz, P.**, **Yilmazer, D.**, and **Isli, D.**, 2015, Veri Yoğun Bilgi Sistemleri için Melez Bir Veri Mimarisi Önerisi., in *UYMS*.

KAYNAKLAR DİZİNİ (devam)

Vajantri, A.K., **Toor, K.D.S.**, **Begoli, E.**, and **Bates, J.**, 2017, An apache calcite-based polystore variation for federated querying of heterogeneous healthcare sources, in *2017 IEEE International Conference on Big Data (Big Data)*, pages 3221–3227, IEEE.

Vogt, M., **Hansen, N.**, **Schönholz, J.**, **Lengweiler, D.**, **Geissmann, I.**, **Philipp, S.**, **Stiemer, A.**, and **Schuldt, H.**, 2021, Polypheny-DB: towards bridging the gap between polystores and HTAP systems, in *Heterogeneous Data Management, Polystores, and Analytics for Healthcare: VLDB Workshops, Poly 2020 and DMAH 2020, Virtual Event, August 31 and September 4, 2020, Revised Selected Papers 6*, pages 25–36, Springer.

Vogt, M., **Stiemer, A.**, and **Schuldt, H.**, 2018, Polypheny-DB: towards a distributed and self-adaptive polystore, in *2018 IEEE International Conference on Big Data (Big Data)*, pages 3364–3373, IEEE.

Wang, J., **Baker, T.**, **Balazinska, M.**, **Halperin, D.**, **Haynes, B.**, **Howe, B.**, **Hutchison, D.**, **Jain, S.**, **Maas, R.**, **Mehta, P.**, **et al.**, 2017a, The Myria Big Data Management and Analytics System and Cloud Services., in *CIDR*.

Wang, L., **Zhang, W.**, **He, X.**, and **Zha, H.**, 2018, Supervised reinforcement learning with recurrent neural network for dynamic treatment recommendation, in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2447–2456.

Wang, M., **Zhang, J.**, **Liu, J.**, **Hu, W.**, **Wang, S.**, **Li, X.**, and **Liu, W.**, 2017b, Pdd graph: Bridging electronic medical records and biomedical knowledge graphs via entity linking, in *The Semantic Web–ISWC 2017: 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part II 16*, pages 219–227, Springer.

KAYNAKLAR DİZİNİ (devam)

Wang, S., **McDermott, M.B.**, **Chauhan, G.**, **Ghassemi, M.**, **Hughes, M.C.**, and **Naumann, T.**, 2020, Mimic-extract: A data extraction, preprocessing, and representation pipeline for mimic-iii, in *Proceedings of the ACM conference on health, inference, and learning*, pages 222–235.

Wieringa, R.J., 2014, *Design science methodology for information systems and software engineering*, Springer.

Wu, M., **Hughes, M.**, **Parbhoo, S.**, **Zazzi, M.**, **Roth, V.**, and **Doshi-Velez, F.**, 2018, Beyond sparsity: Tree regularization of deep models for interpretability, in *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Yang, X., **Bian, J.**, **Hogan, W.R.**, and **Wu, Y.**, 2020, Clinical concept extraction using transformers, *Journal of the American Medical Informatics Association*, 27(12), 1935–1942.

Yu, K., **Gadepally, V.**, and **Stonebraker, M.**, 2017, Database engine integration and performance analysis of the BigDAWG polystore system, in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, IEEE.

Zhang, A., **Xing, L.**, **Zou, J.**, and **Wu, J.C.**, 2022, Shifting machine learning for healthcare from development to deployment and from models to data, *Nature Biomedical Engineering*, 6(12), 1330–1345.

Zhang, Y., **Zhang, Y.**, **Qi, P.**, **Manning, C.D.**, and **Langlotz, C.P.**, 2021, Biomedical and clinical English model packages for the Stanza Python NLP library, *Journal of the American Medical Informatics Association*, 28(9), 1892–1899.

Zheng, X., **Dasgupta, S.**, **Kumar, A.**, and **Gupta, A.**, 2021, AWESOME:

KAYNAKLAR DİZİNİ (devam)

Empowering Scalable Data Science on Social Media Data with an Optimized Tri-Store Data System, *arXiv preprint arXiv:2112.00833*.

Zheng, X., **Dasgupta, S.**, **Kumar, A.**, and **Gupta, A.**, 2023, An Optimized Tri-store System for Multi-model Data Analytics, *arXiv preprint arXiv:2305.14391*.

TEŞEKKÜR

Doktora öğrenimim boyunca uzmanlığı, kıymetli fikirleri, rehberliği ve desteği ile tezin tamamlanmasını sağlayan, lisanstan bu yana değerli danışmanım sayın Doç. Dr. Rıza Cenk ERDUR'a özel olarak teşekkür ederim.

Vizyonu, ileri görüşlülüğü, derin teknik ve akademik bilgi birikimi ile hem sürekli öğrenen hem de teknik ve mental anlamda öğretmeye ve ufkumu genişletmeye devam eden duayen hocam ve ikinci danışmanım sayın Prof. Dr. Oğuz DİKENELLİ'ye sonsuz teşekkür ederim.

Tez izleme komitesinde yer alarak; fikirleri, görüşleri ve önerileri ile tezi ilerletmemde bana destek olup katkı sunan değerli hocalarım sayın Doç. Dr. Övünç ÖZTÜRK'e ve sayın Dr. Öğr. Üyesi Özgür GÜMÜŞ'e içten teşekkürlerimi sunarım.

Çetin ve yorucu geçen bu süreçte vermiş oldukları koşulsuz destek ile güçlükleri aşabildiğim, maddi ve manevi her daim yanımda olan sevgili eşim Selay SERT YÖNYÜL'e ve kıymetli aileme çok teşekkür ederim.

Ayrıca Seagent laboratuvarı başta olmak üzere, bu yolda omuz omuza yürüdüğüm çalışma arkadaşlarıma ve dostum Ramadan ALTINKAYA'ya teşekkürü bir borç bilirim.

14/09/2023

İmzası

Burak YÖNYÜL

ÖZGEÇMİŞ

Kişisel Bilgiler

Burak YÖNYÜL

Gaziemir, İzmir

Eğitim Bilgileri

- (Lisans), Ege Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği, (2006-2011)
- (Yüksek Lisans), Ege Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği, (2011-2014)
- (Doktora), Ege Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği, (2014-2023)

Çalışma Deneyimi

- Stajyer, Türk Telekomünikasyon A.Ş., İzmir (01.07.2010-31.07.2010)
- Bursiyer Öğrenci, Seagent Laboratuvarı - TÜBİTAK Projesi, İzmir (01.09.2011-12.01.2014)
- Araştırma - Yazılım Geliştirme, Galaksiya Bilişim Teknolojileri ve Danışmanlık LTD. ŞTİ. (03.02.2014-24.09.2014)
- Araştırma Görevlisi, Mehmet Akif ERSOY Üniversitesi Mühendislik-Mimarlık Fakültesi Bilgisayar Mühendisliği Bölümü, İzmir (17.02.2016-24.05.2016)
- Öğretim Görevlisi, Ege Üniversitesi Mühendislik Fakültesi Bilgisar Mühendisliği Bölümü, İzmir (15.05.2017-)

Çalışma Alanları

- Çoklu Depo Sistemleri
- Ölçeklenebilir ve Paralel Sorgu İşleme
- Büyük Veri Yönetimi
- Bağlı Veri ve Anlamsal Web