



**BÜYÜK VERİ İŞLEME UYGULAMALARI İÇİN BİR LAMBDA MİMARİ
GELİŞTİRİLMESİ**

Mustafa Umut DEMİREZEN

**DOKTORA TEZİ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI**

**GAZİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ**

ARALIK 2015

Mustafa Umut DEMİREZEN tarafından hazırlanan “BÜYÜK VERİ İŞLEME UYGULAMALARI İÇİN BİR LAMBDA MİMARİ GELİŞTİRİLMESİ” adlı tez çalışması aşağıdaki jüri tarafından OY BİRLİĞİ / OY ÇOKLUĞU ile Gazi Üniversitesi Elektrik-Elektronik Mühendisliği Anabilim Dalında DOKTORA TEZİ olarak kabul edilmiştir.

Danışman: Yrd.Doç.Dr. Tuğba Selcen NAVRUZ

Elektrik-Elektronik Mühendisliği, Gazi Üniversitesi

Bu tezin, kapsam ve kalite olarak Doktora Tezi olduğunu onaylıyorum/onaylamıyorum

Başkan : Prof.Dr. Şeref SAĞIROĞLU

Bilgisayar Mühendisliği, Gazi Üniversitesi

Bu tezin, kapsam ve kalite olarak Doktora Tezi olduğunu onaylıyorum/onaylamıyorum

Üye : Prof.Dr. Cengiz TAPLAMACIOĞLU

Elektrik-Elektronik Mühendisliği, Gazi Üniversitesi

Bu tezin, kapsam ve kalite olarak Doktora Tezi olduğunu onaylıyorum/onaylamıyorum

Üye : Prof.Dr. Erdoğan DOĞDU

Bilgisayar Mühendisliği, TOBB Ekonomi ve Teknoloji Üniversitesi

Bu tezin, kapsam ve kalite olarak Doktora Tezi olduğunu onaylıyorum/onaylamıyorum

Üye : Yrd.Doç.Dr. Atila BOSTAN

Bilgisayar Mühendisliği, ATILIM Üniversitesi

Bu tezin, kapsam ve kalite olarak Doktora Tezi olduğunu onaylıyorum/onaylamıyorum

Tez Savunma Tarihi: 18/12/2015

Jüri tarafından kabul edilen bu tezin Doktora Tezi olması için gerekli şartları yerine getirdiğini onaylıyorum.

.....
Prof. Dr. Metin GÜRÜ
Fen Bilimleri Enstitüsü Müdürü

ETİK BEYAN

Gazi Üniversitesi Fen Bilimleri Enstitüsü Tez Yazım Kurallarına uygun olarak hazırladığım bu tez çalışmada;

- Tez içinde sunduğum verileri, bilgileri ve dokümanları akademik ve etik kurallar çerçevesinde elde ettiğimi,
- Tüm bilgi, belge, değerlendirme ve sonuçları bilimsel etik ve ahlak kurallarına uygun olarak sunduğumu,
- Tez çalışmada yararlandığım eserlerin tümüne uygun atıfta bulunarak kaynak gösterdiğimi,
- Kullanılan verilerde herhangi bir değişiklik yapmadığımı,
- Bu tezde sunduğum çalışmanın özgün olduğunu,

bildirir, aksi bir durumda aleyhime doğabilecek tüm hak kayıplarını kabullendiğimi beyan ederim.

M. Umut DEMİREZEN

18/12/2015

BÜYÜK VERİ İŞLEME UYGULAMALARI İÇİN BİR LAMDA MİMARİ GELİŞTİRİLMESİ

(Doktora Tezi)

Mustafa Umut DEMİREZEN

GAZİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

Aralık 2015

ÖZET

Büyük Veri teknolojileri ilk olarak kullanılmaya başlandığından beri ihtiyaç duyulan amaçlar doğrultusunda değişim geçirmiştir. Başlangıçta sadece duran veri işleme sistemleri kullanılıyorken zamanla akışkan verilerin de işlenebilmesi ihtiyacı doğmuştur. Günümüzde ise büyük verilerden fayda sağlanabilmesi için bu ihtiyaç, duran ve hareketli verilerin birlikte işlenebilmesini gerektirmektedir. Bu konuda halen olgunlaşmış bir teknoloji olmasa da teorik ve konsept bazında yayınlanan çalışmalar ilgi çekicidir. Bu çalışmada henüz konsept bazında, akademik olarak yeterince araştırılmamış ve pratikte yaygın olarak kullanılmayan yeni bir Büyük Veri teknolojisi olan Lamda Mimari üzerinde detaylı bir araştırma ve geliştirme faaliyeti icra edilmiştir. Üç farklı katmandan oluşan mimarinin akışkan veri, duran veri ve sunum katmanlarının, konseptin tarif ettiği şekilde tam olarak kullanılmasıyla özgün bir tasarım geliştirilmiştir. Özellikle sunum katmanında mimarinin ihtiyaçlarını karşılayacak bir teknolojinin mevcut olmaması nedeniyle bu konuya odaklanılarak, mimarinin ihtiyaç duyduğu şekilde bir katman oluşturulabilmiştir. Ayrıca duran ve akışkan veri katmanlarında gerçekleştirilecek farklı karakteristik ve nitelikteki Büyük Veri analiz işlemleri için gerekli yazılım/kod üretiminin tek bir platformda toplanması sağlanarak yeni bir mimari oluşturulmuştur. Bu mimariye gerçek zamanlı uygulama katmanı da eklenerek orijinal konseptinde olmayan bir özellik kazandırılmıştır. Gerçekleştirilen testler sonucunda tasarımın deklare edilen konseptinde olmayan yeni yetenekleri de bünyesinde barındıracak şekilde, gerçek zamanlı ve durağan verilerin birlikte işlenebildiği özgün bir Lamda Mimari tasarlanmıştır. Gerçek hayatta karşılaşılabilecek senaryolar doğrultusunda yürütülen yedi farklı testin sonucunda tasarımın Lamda Mimari olarak oldukça güvenilir olarak çalıştığı tespit edilmiştir.

Bilim Kodu : 905.1.1014
Anahtar Kelimeler : Büyük Veri, Lamda Mimari
Sayfa Adedi : 132
Danışman : Yrd. Doç. Dr. Tuğba Selcen NAVRUZ

DEVELOPING A LAMBDA ARCHITECTURE FOR BIG DATA PROCESSING
APPLICATIONS
(Ph. D. Thesis)

Mustafa Umut DEMİREZEN

GAZİ UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
December 2015

ABSTRACT

Big data technologies have had a need for change in line with the objectives needed to be used since the time they first began. Initially only batch processing systems had been used, but the need for real time data processing use also came into being in time. Today, however, to benefit from big data requires for this need a combination of batch and speed layers must be processed together. Even though there has been no matured technology, in terms of research on this issue published works are interesting in relation to theoretical and conceptual framework. In this research, adequate academic research has not been conducted on the basis of concept yet; there hasn't been a satisfactorily detailed and developed research on the lambda architecture, which is a big data technology being unused practically and commonly. When the three-folded batch layer, speed layer and batch processing of the structure are used as indicated by the concept, an original design has been advanced. Especially because of the absence of a technology that meets the needs of serving layer architecture, a layer could have been formed as required by the architecture by focusing on these issues. By collecting on the same platform the production of design and code required for various features and characteristics of batch and real time layers that are needed for the large date analyses, quite an original architecture has been established. By adding a real time layer to this architecture, a new feature that does not exist in its original concept has been gained. As a result of tests, by adding new capabilities to its structure in the declared concept of the design, original lambda architecture has been designed in which real time and batch layer can be processed together. At the end of seven different tests in the direction of scenarios that can be faced in real life, it has been identified that as an architect of lambda, the design has been pretty reliable.

Science Code : 905.1.1014
Key Words : Big Data, Lambda Architecture
Page Number : 132
Supervisor : Assist. Prof. Dr. Tuğba Selcen NAVRUZ

TEŞEKKÜR

Yardımlarını her konuda esirgemeyen sayın danışman hocam Yrd.Doç.Dr. Tuğba Selcen NAVRUZ'a, akademik yaşamımın başlangıcından beri her konuda destek, yardım, iyi niyetini ve ilmini benimle paylaşan değerli hocam sayın Doç.Dr. Timur AYDEMİR'e, bu tez çalışması hakkında yorumlarını, fikirlerini ve katkılarını sunan sayın hocam Prof.Dr. Şeref SAĞIROĞLU ve Yrd.Doç.Dr. Atilla BOSTAN'a, bu günlere gelmemde büyük pay sahibi olan, hayatımın her aşamasında kolaylık, rahatlık ve huzur sağlayan ve eğitim hayatım süresince beni yalnız bırakmayan, desteklerini esirgemeyen ailem Öğr.Grv. Ceylan TÜRKSİN'e, Nezahat TÜRKSİN'e ve ne yazık ki bu çalışmamı tamamlayamadan aramızdan ayrılan sevgili büyük babam (Em.) İs.Alb. Necmettin TÜRKSİN'e teşekkürü bir borç bilirim.

İÇİNDEKİLER

	Sayfa
ÖZET	iv
ABSTRACT.....	v
TEŞEKKÜR.....	vi
İÇİNDEKİLER	vii
ÇİZELGELERİN LİSTESİ.....	x
ŞEKİLLERİN LİSTESİ.....	xi
RESİMLERİN LİSTESİ.....	xiii
SİMGELER VE KISALTMALAR.....	xiv
1. GİRİŞ	1
2. BÜYÜK VERİ VE BÜYÜK VERİ İŞLEME MİMARİLERİ	9
2.1. Büyük Veri Tanımı ve Literatür Özeti	10
2.2. Duran Veri İşleme Modeli	21
2.3. Akışkan Veri İşleme Modeli	23
2.4. Hibrit Veri İşleme Modeli	23
2.5. Büyük Veri İşleme Teknolojileri	25
2.5.1. Duran veri işleme teknolojileri	28
2.5.2. Akışkan veri işleme teknolojileri	32
2.5.3. Hibrit veri işleme teknolojileri	37
3. LAMDA MİMARİ VE TASARIMI	39
3.1. Lamda Mimari	39
3.2. Lamda Mimari Katmanları	40
3.2.1. Duran veri katmanı	42
3.2.2. Akışkan veri katmanı	43

	Sayfa
3.2.3. Sunum katmanı	45
3.3. Lamda Mimari Tasarımı	45
3.3.1. Veri yolu (Data Bus) katmanı	47
3.3.2. Duran veri katmanı	49
3.3.3. Akışkan veri katmanı	51
3.3.4. Sunum katmanı	53
3.3.5. Yönetim/Koordinasyon katmanı	61
3.3.6. Gerçek zamanlı uygulama	64
4. LAMDA MİMARİ GELİŞTİRİLMESİ VE TEST EDİLMESİ	69
4.1. Veri Temini	73
4.1.1. ADS-B sinyalleri	75
4.1.2. ADS-B sinyallerinden veri seti oluşturulması	78
4.1.3. Veri modeli ve serileştirme	82
4.2. Lamda Mimari Bileşenleri	84
4.2.1. Veri yolu (Data Bus) katmanı	84
4.2.2. Duran veri katmanı	86
4.2.3. Akışkan veri katmanı	88
4.2.4. Sunum katmanı	90
4.2.5. Yönetim/koordinasyon katmanı	93
4.2.6. Sistem izleme alt yapısı	94
4.2.7. Test/doğrulama altyapısı (Ground Truth)	95
4.3. Tasarlanan Lamda Mimarinin Test Edilmesi	97
4.3.1. Azami kapasitede çalışma testi	99
4.3.2. Normal kapasite çalışma testi	100

Sayfa

4.3.3. Tekrarlı veri gelmesi ve Lamda Mimarinin davranışının incelenmesi	104
4.3.4. Veri gecikmesi ve lamda mimarinin davranışının incelenmesi	108
4.3.5. Duran veri katmanının çalışmaması durumunda Lamda Mimari davranışının incelenmesi	111
4.3.6. Duran veri katmanının çalışmaması ve tekrarlı veri gelmesi durumunda Lamda Mimari davranışının incelenmesi	113
4.3.7. Duran veri katmanının çalışmaması ve veri gecikmesi durumunda Lamda Mimarinin davranışının incelenmesi	116
5. SONUÇ	121
KAYNAKLAR	125
ÖZGEÇMİŞ	131

ÇİZELGELERİN LİSTESİ

Çizelge	Sayfa
Çizelge 4.1. Lamda Mimari donanım altyapısı	72
Çizelge 4.2. Toplanan verilerden oluşturulan veri setleri	80
Çizelge 4.3. Veri boyut değişkenleri ve uygulanan işlemler	91
Çizelge 5.1. Lamda Mimari akademik çalışma karşılaştırma sonuçları	123

ŞEKİLLERİN LİSTESİ

Şekil	Sayfa
Şekil 2.1. Büyük Veri işleme yaşam döngüsü	24
Şekil 2.2. Büyük Veri işleme teknolojilerinin gelişimi	27
Şekil 3.1. Lamda mimari mantıksal yapı	40
Şekil 3.2. Sistem Mimari ve bileşenleri	46
Şekil 3.3. Veri üretimi yazılımı	48
Şekil 3.4. Druid sorgu ve veri ithal işlemleri	59
Şekil 3.5. Druid yönetim katmanı	59
Şekil 3.6. AVK ve DVK için örnek bir veri işleme akış hızı - zaman grafiği	62
Şekil 4.1. ADS-B ModeS sinyali modülasyonu	71
Şekil 4.2. ADS-B Mesaj formatları için sınıf diyagramları	83
Şekil 4.3. Veri Akış Hızı - Zaman değişimi	93
Şekil 4.4. Sistem izleme alt yapısı	95
Şekil 4.5. Test/Doğrulama alt yapısı	96
Şekil 4.6. Performans testi sonucu veri işleme hızının zamana göre değişimi	100
Şekil 4.7. Normal çalışma durumunda veri işleme hızının zamana göre değişimi	101
Şekil 4.8. Normal çalışma durumu geri getirme ve kesinlik değerlerinin değişimi	102
Şekil 4.9. 01-30 Eylül 2015 tarihleri arası 1 dk'lık zaman penceresi için analiz sonuçları	103
Şekil 4.10. 01-30 Eylül 2015 tarihleri arası 10 dk'lık zaman penceresi için analiz sonuçları	103
Şekil 4.11. 01-30 Eylül 2015 tarihleri arası 1 saatlik zaman penceresi için analiz sonuçları	104
Şekil 4.12. Tekrarlı veri gelmesi durumunda veri işleme hızının zamana göre değişimi	105
Şekil 4.13. Tekrarlı veri gelme durumu geri getirme ve kesinlik değerlerinin değişimi	107

Şekil	Sayfa
Şekil 4.14. Tekrarlı veri gelme durumu geri getirme ve kesinlik değerlerinin değişimi	108
Şekil 4.15. Verinin gecikmesi durumunda veri işleme hızının zamana göre değişimi ..	109
Şekil 4.16. Veri gecikmesi durumunda geri getirme ve kesinlik değerlerinin 15 Eylül 2015 10:00 ile 16 Eylül 2015 14:00 zaman aralığı için değişimi	110
Şekil 4.17. DVK çalışmadığı durumda veri işleme hızının zamana göre değişimi	111
Şekil 4.18. DVK çalışmadığı durumda veri işleme hızının zamana göre değişimi	113
Şekil 4.19. DVK çalışmadığı ve tekrarlı veri geldiği durumda veri işleme hızının zamana göre değişimi	114
Şekil 4.20. DVK çalışmadığı ve tekrarlı veri gelmesi durumunda geri getirme ve kesinlik değerlerinin değişimi	115
Şekil 4.21. DVK çalışmadığı durumda tekrarlı olarak gelen tarihlerde geri getirme ve kesinlik değerlerinin değişimi	116
Şekil 4.22. DVK'nın çalışmadığı ve veri gecikmesi durumunda veri işleme hızının zamana göre değişimi	117
Şekil 4.23. DVK'nın çalışmadığı ve veri gecikmesi durumunda geri getirme ve kesinlik değerlerinin 15 Eylül 2015 10:00 ile 16 Eylül 2015 14:00 zaman aralığı için değişimi	119

RESİMLERİN LİSTESİ

Resim	Sayfa
Resim 3.1. Canlı veri gösterim ara yüzü	66
Resim 3.2. Kelime bulutu gösterimi	66
Resim 3.3. Canlı ısı haritası (HeatMap) gösterimi	67
Resim 3.4. Hava taşıtlarının canlı olarak radar grafiği olarak gösterimi	67
Resim 3.5. Hava taşıtı canlı liste gösterimi	68
Resim 3.6. Hava taşıtı bilgilerin yeniden oynatılması (replay)	68
Resim 4.1. Veri toplama sistemi bina içi yerleşimi	73
Resim 4.2. Veri toplama sistemi bina dışı kurulumu	74
Resim 4.3. Yazılım Tabanlı Telsizlerden elde edilen ham veri	74
Resim 4.4. ETL işleminden sonra analitik olarak işlenecek veriler	84

SİMGELER VE KISALTMALAR

Bu çalışmada kullanılmış simgeler ve kısaltmalar, açıklamaları ile birlikte aşağıda sunulmuştur.

Simgeler

Açıklamalar

s	Saniye
dk	Dakika
Sa	Saat
MB	Mega Byte
GB	Giga Byte
TB	Terra Byte
PB	Peta Byte

Kısaltmalar

Açıklamalar

ADS-B	Automatic Dependent Surveillance – Broadcast
AV	Akışkan Veri
AVK	Akışkan Veri Katmanı
CSV	Comma Separated Values
DV	Duran Veri
DVK	Duran Veri Katmanı
DVG	Duran Veri Görüntüleri (Batch Views)
ETL	Extract Transform Load
GZG	Gerçek Zaman Görüntüleri (Real Time Views)
HDFS	Hadoop Distributed File System
ICAO	International Civil Aviation Organization
IoT	Internet Of Things
JSON	Javascript Object Notation
LM	Lamda Mimari (Lambda Architecture)
NTP	Network Time Protocol
RDD	Resilient Distributed Dataset

Kısaltmalar**Açıklamalar****SK**

Sunum Katmanı

SİA

Sistem İzleme Altyapısı

TDA

Test ve Doğrulama Altyapısı

TSV

Tab Seperated Values

YKK

Yönetim ve Koordinasyon Katmanı

XML

Extensible Markup Language

1. GİRİŞ

Büyük Veri teknolojilerinin popülerliğinin artması ve açık kaynaklı olarak geliştirilmesi nedeniyle birçok programcı bu alanda çalışmalar yapmaya başlamış ve bu teknolojiler birer ekosistem haline gelerek kullanım alanlarını genişletmiştir. Bir biri ardına geliştirilen bu teknolojiler; kişilerin, kurumların ve şirketlerin, verilerin çok büyük olması nedeniyle daha önce gerçekleştiremedikleri iş analitiği, veri analizi ve veri madenciliği gibi alanlarda bu yeteneği kazanmalarını, işlerini kolaylaştırmasını ve daha önce gerçekleştiremedikleri analizleri yapabilir hale gelmesini sağlamıştır. Hatta daha da öteye giderek veri biliminin popüler olması ve yeni bir meslek olarak kabul edilmesine yol açmış ayrıca büyük verilerin incelenerek geleceğe dair kestirim ve analizlerin yapılabilmesine olanak sağlamıştır.

Büyük Veri kavramı içerisinde verinin büyüklüğünden çok yapısı önem arz etmektedir. Büyük Veriyi ortaya çıkaran en temel özellikleri veri hacminin büyük olması, belirli bir yapısallığın olmaması, hızlı şekilde artması ve farklı biçimlerde karşımıza çıkması olarak belirtilebilir. Büyük Veri alanında analizler farklı tipte veri türleri üzerinde gerçekleştirilmektedir. Yapı olarak düşünüldüğünde veriler; yapısız (unstructured), yarı-yapılandırılmış (semi structured) veriler ve yapılandırılmış (structured) veriler olarak gruplanabilirler. Ayrıca veri işleme yöntemine göre gruplandığında ise duran veri (Data at Rest), akışkan veri (Data in Motion), bağlı veri (Linked-Data) ve zaman serisi (Time Series Data) olarak da sınıflandırılabilirler. Farklı türdeki verilerin işlenebilmesi için de farklı yöntemlerin geliştirilmesi gerekmektedir. Dolayısı ile Büyük Veri sistemleri ve işleme yöntemleri de bu doğrultuda farklılık göstermektedir. Büyük Veri işleyebilme ihtiyacı zaman içerisinde karşılaşılan yeni veri türleri doğrultusunda şekillenmiştir.

Duran veri olarak sınıflandırılan veriler, üretildikten sonra belirli bir yere kayıt edilen ve saklanan verileri temsil etmektedir. Bu veri türü elde edildikten sonra değişmemekte ve toplu olarak (batch) saklanmak suretiyle işlenmektedir. Hareketli veriler ise sürekli olarak üretildiği kaynaktan başka bir sisteme belirli bir akış hızı ve büyüklükle transfer edilen, gerçek zamanlı ve yakın gerçek zamanlı biçimde işlenerek anlam kazandırılması gereken verilerdir. Bu veri tipine en iyi örnek olarak Nesnelerin İnterneti (IoT) cihazlarının ürettiği sensör verileri gösterilebilir. Sosyal medyada üretilen, birbirleri ile ilişkili olarak tanımlanan ve çizge (graph) yöntemleri ile modellenebilen veri türleri ise bağlı veri olarak

tanımlanmaktadır. Zamana göre deęişim gösteren ve sensör tabanlı ölçümler sonucunda elde edilen veriler ise zaman serisi verileri olarak sınıflandırılmaktadır.

Bu tanımlamalar ve sınıflandırmalar ışığında, her bir veri türünün incelenmesi için farklı çözümler üretilmesi gerekmektedir. Hareketli bir verinin, duran bir veri olarak işlenmesinin mümkün olmaması gibi tersi de mümkün değildir. İşleme yöntemleri arasında dönüşüm yapılabilmesi mümkün olsa da dönüşüm işleminin gerçekleştirilmesi ve buna zaman ve kaynak ayrılması mantıklı bir çözüm değildir. Ayrıca hareketli bir veri duran bir veri türüne dönüştürülebilse de, verinin işlenmesi ve kıymetlendirilmesinin gerçek zamanlı ya da yakın gerçek zamanlı olarak tamamlanması gerekmektedir. Bu gibi teknik gereksinimler nedeniyle farklı metotlar ile işlenmesi gereken veriler için farklı yöntemler geliştirilmiştir. Bu ihtiyaçtan doğan açık kaynaklı Büyük Veri projelerinin sayısı her geçen gün artmaktadır. Söz konusu alanda çalışma yürüten gönüllü yazılımcılar ya da şirketler birbiri ardına farklı yazılım çatıları, platformlar ve alt yapılar geliştirmektedir. Bunlar sadece belirli bir süre ilgi çekmekte ve kullanılmaktadır. Ancak daha sonra yeni bir teknolojinin yani veri işleme yönteminin ortaya çıkarılması sonucu popülerliğini yitirmektedir. Çoğu şirket kullandığı teknolojiyi, yeni çıkan teknoloji ile değiştirememektedir. Veri büyüklüğünün tera/peta byte'lar seviyesinde olduğu düşünüldüğünde, yeni teknolojinin olgunlaşmasından emin olunmadan değişikliğe gitmek ve bu alanda yatırım yapmak çok doğru bir yaklaşım olmayacaktır.

Teknoloji geliştiren yazılımcıların mevcut teknolojilerin üzerine farklı katmanlar geliştirerek veri işleme yöntemini değiştirdiği durumlar da gözlenmektedir. Özellikle gerçek uygulamalarda yaşanan sorunlar ve sıkıntılar nedeniyle bunlardan ders alınarak farklı yaklaşımlar sergilenmektedir. Örnek olarak, bu alanda ilk geliştirilen teknolojilerden biri olan ve duran veri işlemede kullanılan Apache Hadoop projesi birkaç yıl içerisinde karşılaşılan sorunlar nedeniyle Apache Hadoop 2.0. YARN olarak geliştirilmeye başlanmıştır. Bu yeni yaklaşımla beraber kaynak yönetiminde iyileştirilmelere gidilerek tak-çalıştır bir mimariye geçilmiştir. Hadoop ekosisteminde geliştirilen her bir proje, YARN'ın kaynak yönetimi ve YARN'ın sağlamış olduğu uygulama programlama arayüzleri (API) sayesinde kolayca bu yeni kaynak yönetimi sisteminden faydalanabilmiş ve alt yapısal karmaşıklıktan kendilerini soyutlamayı başarmıştır. Bugün bu teknoloji oldukça olgunlaştığı için birçok alanda kullanılmaya devam edilmektedir. Çeşitli şirketler bu alt yapıyı kullanarak kendilerine uygun daha üst seviye veri işleme kabiliyetleri ve

teknolojileri geliştirmektedirler. Örnek olarak Facebook'un geliştirdiği HIVE projesi Hadoop üzerinde bulunan duran verilerin SQL diliyle sorgulanmasını sağladığı gibi, Pig projesi ile Yahoo firması kendine özel bir betik dili olan Pig-Latin'i geliştirmiş ve kullanmaya başlamıştır. Daha sonra söz konusu iki proje de açık kaynak kodlu olarak yayınlanmıştır. Bugün hala bu projeler aktif olarak kullanılmaktadır. Temel olarak bu teknolojilerin yaptığı işlem SQL ve Pig-Latin dillerinde yazılan kodların Map-Reduce işlerine (job) dönüştürülerek analiz yapılmasına olanak sağlamasıdır.

Zaman içerisinde Google'ın ortaya attığı Map-Reduce yöntemi ile veri işlenmesinden çok daha hızlı çalışan, performans ve küme (cluster) kaynakları kullanım performansı daha iyi olan, Cloudera firmasının geliştirdiği Tez projesine başlanmıştır. Bu teknoloji ile aynı miktardaki Büyük Veri Hive ve Pig projelerine göre daha hızlı işlenebilmektedir.

Haraketli veri işleme alanında ise Twitter, LinkedIn gibi büyük ölçekli firmalar kendi teknolojilerini kendi ihtiyaçları doğrultusunda geliştirmiş, daha sonra açık kaynaklı yazılım olarak kullanıma sunmuştur. Hareketli veri belki de işlenmesi en zor veri türüdür. Anlık olarak elde edilen verilerin anlamlandırılarak ilgili birime gönderilmesi, sunulması ve görselleştirilmesi oldukça zor bir işlemdir. Özellikle yüksek hızlarda ve hacimde akan bu verilerin anlık olarak işlenebilmesi özel bir teknoloji gerektirmektedir. Bu alanda kullanılan popüler teknolojiler Twitter tarafından geliştirilen Apache Storm, LinkedIn firmasında geliştirilen Apache Samza ve Berkeley tarafından geliştirilen Spark'tır. Her birinin özellikleri, yapısal birimleri, çalışma şekilleri ve performansları farklı olsa da özellikle hareket halindeki verileri yatay olarak ölçeklenebilir işleyebilmek için geliştirilmişlerdir. Gerçek zamanlı olarak ya da yakın gerçek zamanlı olarak işlenmesi gereken veriler bu teknolojiler ile kıymetlendirilerek anlık olarak görselleştirilir ve anlamlandırılabilirler.

Büyük Veri işleme altyapıları ve sistemleri sayesinde çıkart-transfer et-yükle (ETL) işlemleri gerçekleştirilebildiği gibi verinin işlemeye elverişli hale getirilmesi de sağlanmaktadır. Tip ve yapı değiştiren bu veriler iş analitiği ve raporlama uygulamalarında kullanıldığı gibi kestirim analitiği (predictive analytics) uygulamalarında da kullanılmaktadır. Dolayısı ile Büyük Veri teknolojileri ve projeleri aslında birer araçtır. Esas amaç ise veriyi kıymetlendirerek bunlardan bir ürün elde etmektir. Bu işlemler ise Büyük Veri teknolojileri kullanılarak veri bilimi metotları ile yapılmaktadır. Veri ile

alıřan, veri reten tm kuruluřlar veri bilimi sayesinde birok konuda bu alandan faydalanmaktadır. Byk Veri ekosistemi projeleri de veri bilimi iin teknolojiler geliřtirmiřtir. Apache Mahout, Apache Spark MLLib gibi projeler makine ğrenimi algoritmalarının byk veriler ile kullanılmasına olanak saėlamaktadır.

Her geen gn yeni bir byk veri iřleme teknolojisinin ve ynteminin geliřtirilmesine bařlanması ve birden ok projenin bir arada kullanılabilmesi nedeniyle belirli teknolojiler evresinde ekosistemler oluřmaktadır. Bu alanda alıřmalar yrten ve teknoloji geliřtiren řirketler ise ihtiya duyan firmalar ve kurumlar iin birden ok ekosistemi bir araya getirerek zel daėıtımlar oluřturmaktadırlar. Cloudera, HortonWorks ve MapR gibi bu alanda sz sahibi olan ve teknolojiye yn veren řirketler, kendi geliřtirdikleri bazı yeni teknolojilerini aık kaynak olarak yayınlamamaktadır. Ayrıca mevcut ekosistem projelerini kendilerine gre bařka alt yapılar, programlama dilleri ile yeniden geliřtirerek, aık kaynak kodlu halinden daha iyi zelliklere sahip bir rn olarak sunmaktadırlar. Byk Veri daėıtımı (distribution) reten birok řirket, kendilerine zg Byk Veri projelerini ierisinde barındıran daėıtımlarına, diėer aık kaynaklı ekosistem projelerini de eklemek suretiyle, bu alanda alıřma yapan sadece belirli bir konuya odaklanmış firmaların da yeteneklerini kendi daėıtımları ile birleřtirerek yayınlamaktadır.

Grldė gibi birok aık kaynak proje, teknoloji odaklı ekosistemleri, ekosistemler ise Byk Veri daėıtımlarını meydana getirmektedir. İhtiya sahibi kurum ya da kuruluřlar, daha nce bahsedilen veri tip ve trlerinin iřlenmesine olanak saėlayan teknolojileri iinde barındıran Byk Veri iřleme platformu daėıtımlarını kullanarak, her tipteki Byk Verilerini rahatlıkla iřleyebilmektedir. Yinelemek gerekirse nemli olan, veri bilimi ile veriden faydalanmak ve bu bilimin gerekleřmesine olanak saėlayan Byk Veri teknolojilerini kullanabilmek, veriden elde edilen bilginin eyleme dnřtrlmesini saėlayabilmektir.

Farklı trdeki byk verilerin iřlenmesi maksadıyla ortaya atılan farklı projeler ve teknolojiler olmasına raėmen, aynı anda birden fazla Byk Veri trn iřlemeye olanak saėlayan teknolojiler henz belirli bir olgunluėa ulařmamıř ya da ortaya ıkmamıřtır. Teorik olarak kullanılması uygun olarak dřnlen fikirler gerek hayata geirildiėinde bu alanda kullanılmaya ok elveriřli olmadıėı grlmřtir. Duran ve hareketli verilerin aynı

anda işlenmesi ihtiyacının zaman içerisinde ortaya çıkmasıyla, bu alanda çok olgunlaşmamış bazı fikirler ve yöntemler kavram olarak ortaya atılmaya başlanmıştır.

Daha önce de kullanılan bir yöntem olan, veri türüne göre Büyük Veri ekosistemi bileşenlerinden uygun olanların bir araya getirilmesi ve birlikte kullanılması yaklaşımı, “Lamda Mimari” kavramı olarak tanımlanmaktadır [1]. Lamda Mimari, yapısında duran ve akışkan verilerin aynı anda işlenmesine olanak sağlayan teknolojilerin kullanıldığı bir yaklaşım olup, henüz çok yeni bir kavramdır. Bu alanda farklı yöntemler ve yaklaşımlar denenmiş olsa da henüz teknolojik olarak belirli bir olgunluk seviyesine erişilememiştir.

Özellikle Büyük Veri yaklaşımlarında gerçek zamanlı verilerin işlenmesi ve kullanımı şeklinde bir eğilim gelişmiştir. Veriler kaynağından oluştuğu anda ya da biraz gecikmeli olarak işlendiğinde, biriktirilip sonraki bir zamanda işlendiği duruma göre daha büyük avantaj sağlamaktadır. Verinin oluştuğu anda işlenmeye başlaması ile elde edilen sonuçların hayata geçirilmesi veriden daha erken faydalanılmasını sağlayacaktır. Dolayısı ile veri ne kadar erken işlenirse kazanım da o kadar erken olacaktır. Google’ın MapReduce tabanlı işleme yöntemlerini bırakmasıyla ve gerçek zamanlı olduğu kadar arttırımsal çalışan teknolojilerin geliştirilmesiyle, Büyük Veri işleme yöntemleri bu teknolojilere doğru kayarak popülerlik kazanmıştır. Ancak geçmiş zamanlarda elde edilen ve duran hale getirilen verilerden de faydalanılması gerekmektedir. Tek başına anlık gelen verilerden yeteri kadar faydalanılamamaktadır. Bunun en büyük nedenlerinden birisi veri biliminde model tabanlı sistemler geliştirilirken modelin öğretilmesi için eğitim ve test aşamalarında olabildiğince çok verinin kullanılması gerekliliğidir. Zaman içerisinde geçmişte elde edilen verilerin de incelenmesi ve kullanıma sunulması gerekmektedir.

Eski veriler ve anlık olarak gelen yeni verilerin birlikte işlenmesi ihtiyacı sonucunda Lamda Mimari konsepti ortaya çıkmıştır. Teknolojik olarak henüz bir olgunluğa ulaşmasa da gelecekte kullanılacak en önemli Büyük Veri işleme teknolojisi olacağı değerlendirilmektedir [2]. Lamda Mimarinin en önemli özelliği, işlenecek veri türüne göre sadece o veri türünü işlemek için geliştirilen yöntemlerin ve alt yapıların bir arada kullanılıyor olmasıdır. Dolayısı ile aynı anda hem duran veri işleme hem de akışkan veri işleme yöntemleri kullanılmaktadır. Farklı katmanlar halinde oluşturulan mimari sayesinde akışkan veri katmanından kaynaklanacak sorunlar duran veri katmanı ile düzeltilmektedir.

Böylece sistem eninde sonunda güvenilir (eventually accurate) bir şekilde Brewer's teoremine (CAP teoremine) uygun biçimde çalışmaktadır.

Bu çalışmada daha önce bahsedilen veri türlerinden duran ve akışkan verilerin birlikte işlenmesi ihtiyacına karşı ortaya atılan bir kavramın incelenmesi, bu amaca hizmet edecek bir Lamda Mimarinin gerçekleştirilmesi ve deneysel olarak doğrulanması sağlanmıştır. Teknolojik olarak olgunlaşmamış bir alanda gerçekleştirilen bu çalışma ile yeni bir konseptin en etkin şekilde geliştirilmesi ve test edilmesi amaçlanmıştır. Bu çalışmada özellikle şu ana kadar bilinen yöntemlerin dışında, daha yenilikçi ve etkin bir tasarım hedeflenerek geliştirilen Lamda Mimari tasarımı üzerinde çalışılmıştır.

Bu tasarım, literatürde yayınlanmış hiçbir tasarıma benzememekte olup, Lamda Mimari konseptini tam olarak kullanan ilk tasarımıdır. Özellikle farklı katmanlar için farklı geliştirme yapılması zorunluluğunu ortadan kaldırarak, tek bir defa analiz ve geliştirme yapılmasıyla her iki katmanda da kullanımı ve sunum katmanının tam olarak Lamda Mimari konseptine göre geliştirilmiş olması açısından oldukça özgünlük göstermektedir. Literatürde de görüleceği gibi Lamda Mimarinin sunum katmanı hakkında yeterli inceleme ve araştırma bulunmamaktadır. Çalışmanın sunum katmanı tasarımı, geliştirilmesi ve test edilmesi gibi konulardaki eksiklikleri gidermesi de amaçlanmıştır.

Büyük Veri sisteminin test edilmesi için oldukça fazla miktarda veriye ihtiyaç duyulmaktadır. Bu veriler genellikle ya duran veri ya da akışkan veri türünde olmaktadır. Test işlemleri genellikle açık kaynaklı test verileri ile gerçekleştirilmektedir. Lamda Mimari sisteminin testi için aynı anda iki farklı türde veriye ihtiyaç duyulmaktadır. Lamda Mimarinin duran ve akışkan veri katmanlarında bu verilerin işlenmesi ve sonuçların test edilmesi gerekmektedir. Bilinen veri setlerinin kullanılması yerine, ihtiyaç duyulan veriler, tamamen bu çalışma için Lamda Mimariye uygun, hava taşıtlarından Automatic Dependent Surveillance - Broadcast (ADS-B) sinyallerinin çözülmesi ile elde edilmiştir. Kendi test verisini kendisi üretmesi ve sağlaması açısından da özgün bir çalışma olduğu değerlendirilmektedir.

Bu Tez çalışmasının 2. bölümünde öncelikle literatür taraması ile bu alanda daha önceden beri kullanılan Büyük Veri işleme mimarileri üzerine gerçekleştirilen çalışmalar değerlendirilmiştir. Ayrıca bu bölümde farklı Büyük Veri türlerine değinilerek bunların en

etkin işleme yöntemleri, alt yapıları ve mevcut teknolojiler hakkında bilgiler de sunulmaktadır.

3. Bölümde Lamda Mimari konsepti ve bu konudaki çalışmalar detaylı olarak incelenmiş, özgün bir Lamda Mimari yapısı tasarımının, her bir alt bileşeni içerecek şekilde detaylı ve eksiksiz olarak gerçekleştirilmesi anlatılmıştır. 3. Bölümde literatürde anlatılan Lamda Mimarilerden farklı, yeni bir yaklaşım olarak, Lamda Mimariye gerçek zamanlı uygulama katmanının eklenmesi ve tasarımı da ayrıca açıklanmıştır.

4. Bölümde, 3. Bölümde tasarımı yapılan özgün Lamda Mimarinin, seçilen ve ayrıca geliştirilen alt yapılar ile oluşturulması anlatılmıştır. Bununla birlikte gerçek zamanlı olarak hava taşıtlarından yayınlanan Automatic Dependent Surveillance - Broadcast (ADS-B) sinyallerinin çözümlenmesi ile elde edilen veriler kullanılarak, mimarinin farklı çalışma koşulları altında deneysel olarak doğrulanması ve test edilmesi gerçekleştirilmiştir.

5. Bölümde ise test sonuçları ışığında geliştirilen Lamda Mimarinin kazandırdığı yeni ve özgün yeteneklere değinilerek, literatürdeki diğer Lamda Mimariler ile karşılaştırılmış ve üstün yönleri ortaya konulmuştur. Ayrıca geliştirilen Lamda Mimari test alt yapısının kullanımı ve etkinliği incelenmiştir.

2. BÜYÜK VERİ VE BÜYÜK VERİ İŞLEME MİMARİLERİ

Bu bölümde öncelikle duran ve akışkan veriler ile ilgili veri işleme mimarileri hakkında literatür taraması sonuçları sunulmuştur. Daha sonra duran ve akışkan verilerin özellikleri belirtilerek, bu verilerin işleme yöntemleri üzerinde durulmuştur. Son olarak duran ve akışkan verilerin işlenmesi ile ilgili teknolojiler hakkında bilgi verilmiştir.

Büyük Veri, geleneksel veri tabanları ile kabul edilebilir bir zaman zarfında işlenemeyecek kadar büyük hacimli ve karmaşık yapıda bulunan veri setleri koleksiyonu olarak tanımlanabilir. Geleneksel veri tabanları, yapısal verilerin depolanması ve işlenmesi alanında oldukça etkin olduklarını kanıtlasalar da, çok hızlı, büyük hacimli, çeşitli türde, belirli bir yapısı olmayan veri türleri üzerinde aynı etkinliği gösterememektedirler. Büyük Veri bilimi; mühendislik, sağlık ve finans gibi birçok alanda yaygın olarak kullanılmaktadır. Büyük Veri genellikle 3V ile karakterize edilmektedir. Bunlar çeşitlilik (variety), hacim (volume) ve hız (velocity) olarak belirtilmektedir. Çeşitlilik büyük verinin doğasını ve yapısını oluşturan bilgiyi, hız verinin üretim sıklığını ve dinamizmini, hacim ise verinin büyüklüğünü temsil etmektedir. Özellikle çeşitlilik farklı kaynaklarca oluşturulmuş farklı yapıya sahip verilerin çok farklı şekillerde modellenebilmesini vurgulamaktadır.

Büyük Veri işleme yöntemleri veri türlerine göre farklılık göstermektedir. Verinin hacminden daha önemli olan yapısal karmaşıklığı, formatı ve hareketliliğidir. Bu açıdan bakıldığında farklı veri türlerinin incelenebilmesi için farklı teknolojilerin kullanılması gerekmektedir. Veriler, işleme yöntemi açısından duran veriler ve akışkan veriler olarak iki farklı grupta toplanabilir. Her bir gruptaki verilerin işlenmesi için farklı yaklaşımlar ve yöntemler gerekmektedir. Duyulan ihtiyaca göre, işlenmek istenen veri türüne göre farklı mimarilerin kullanılması gerekmekte ve dolayısı ile birbiri ile aynı olmayan yöntemler ve mimariler geliştirilmektedir.

Büyük Verinin işlenmesi; verinin varlığı ve temini göz önüne alınarak toplu (batch) veri işleme, gerçek zamanlı veri işleme ve hibrit veri işleme olarak sınıflandırılabilir. Toplu veri işleme belirli bir zaman aralığında toplanmış yüksek hacimli verilerin etkin bir şekilde işlenmesi olarak tanımlanmaktadır. Bu işleme yönteminde veriler öncelikle kaynaklarından

toplanmakta, biriktirilmekte ve daha sonra toplu olarak işlenmektedir. Ancak bazı uygulamalar farklı kaynaklardan gelen verilerin gerçek zamanlı olarak işlenmesine ihtiyaç duymaktadır. Gerçek zamanlı veri işleme bir veri kaynağından sürekli olarak gelen verinin işlenerek hemen sonuçlarının elde edilmesini sağlamaktadır. Bu veri işleme yönteminin temel hedefi, ufak parçalar halinde verinin en az gecikme ile işlenmesidir. Hibrit veri işleme birden fazla ya da aynı veri kaynağından gelen duran ve hareketli verilerin, işleme mantığı ve ihtiyaca göre birlikte işlenmesine olanak tanımaktadır.

2.1. Büyük Veri Tanımı ve Literatür Özeti

Büyük Veri genellikle geleneksel veri tabanı ve yazılımlar ile işlenemeyen, büyük boyutlardaki yapısal, yarı-yapısal ve/veya yapısal olmayan veri olarak tanımlanmaktadır [1]. Büyük veriler çeşitli türde verileri içermeleri, yüksek boyutlu olmaları ve yüksek hızlarda üretilmelerinden dolayı yeni yönetim ve işleme metotlarına ihtiyaç duymaktadırlar. Büyük veriler önceleri üç özellik ile karakterize edilirken daha sonraki zamanlarda büyük verinin tanımının da değişmesi sonucunda beş farklı karakteristik özellik ile karakterize edilmeye başlanmıştır:

- **Hacim/Boyut (Volume):** İşlenmesi gereken veri büyüklüğüdür. Büyük Veri geleneksel sistemler ile işlenmesi mümkün olan giga byte, mega byte seviyesinden çok daha büyük olan peta byte ya da daha büyük bir veri boyutuna sahiptir.
- **Hız (Velocity):** Verinin üretim frekansını ve dinamizmini belirtmektedir. Gerçek zamanlı uygulamalarda çok kısa süre içerisinde oldukça fazla veri üretilmesi mümkün olmaktadır.
- **Çeşitlilik (Variety):** Verinin çeşitli yani birden fazla formda ortaya çıkabilmesidir. Dolayısı ile veri farklı kaynaklarda oluşabilmektedir.
- **Güvenilirlik (Veracity):** Verinin güvenilir olmasıdır. Verinin analizlerde doğru sonuca ulaştırabilmesi gerekmektedir. Yüksek hacimli ve farklı kaynaklardan gelen verilerde eksik verilere rastlanabilmektedir.
- **Değer (Value):** Verinin işlemeye değer olmasıdır. Bir analiz için çok kıymetli olan veri başka bir analiz için değersiz olabilmektedir.

Büyük verilerden bir fayda sağlanabilmesi için analizlerinde yeni teknikler ve teknolojiler geliştirilmesi gerekmektedir. Çok disiplinli bu tekniklerin kullanılmasıyla büyük veriler anlamlandırılarak değerli hale getirilebilmektedir. Bu konu üzerine birçok alanda çalışmalar yapılmaktadır. Hali hazırda veri işleme teknikleri olarak toplu veri işleme, gerçek zamanlı veri işleme ve interaktif veri işleme yöntemleri kullanılmaktadır. Veri işleme teknikleri en çok veri madenciliği, makine öğrenimi, yapay sinir ağları, sosyal ağ analizi, sinyal işleme, örüntü tanıma, optimizasyon ve görselleştirme gibi konularda kullanılmaktadır. Büyük veriler üzerine gerçekleştirilmesi gereken analizlerde ilk adım verilerin yapısal hale getirilmesidir. Verilerin ham halde ya da elde edildikleri formda analiz edilebilmesi her zaman mümkün olmamaktadır. Yapısal olmayan verilerin öncelikle analizi kolaylaştıracak şekilde yapısal hale getirilmesi daha sona üzerinde analize başlanması gerekmektedir. Bu nedenle veri işlemeye öncelikle çıkart-dönüştür-yükle (ETL) adımı ile başlanmaktadır.

Toplu veri işleme yöntemleri, büyük verinin boyut (volume) özelliği üzerine odaklanırken, gerçek zamanlı veri işleme yöntemleri verinin hız (velocity) özelliği üzerine odaklanmaktadır. Hibrit veri işleme teknikleri ise büyük ve dinamik yapıda olan büyük verinin hem boyut hem de hız özelliklerine odaklanmaktadır.

İş zekâsı ve analitik uygulamalarında Büyük Veri uygulamaları ile bütünleşmiş olarak çalışma ihtiyacı giderek artmaktadır. Verinin boyutu büyüdükçe, işlenebilmesi ve saklanabilmesi de doğru orantılı olarak zorlaşmaktadır. Geleneksel iş zekâsı uygulamaları bu konuda yetersiz kalmaktadır [2]. Bulut bilişim teknolojilerinin değişmesiyle birçok kuruluş iş zekâsı uygulamalarını sınırsız işlem kabiliyeti sunan bulut bilişim sistemlerine aktarmaktadır [3-5]. Günümüzde MapReduce gibi dağıtık hesaplama yazılım çatıları büyük verilerin işlenmesine olanak sağlamaktadır. MapReduce binlerce düğümlük bir kümeleme sisteminde verilerin dağıtık ve paralel olarak işlenmesine olanak sağlayarak analitik analizler sonunda üretilen sonuçların elde edilmesini mümkün kılmaktadır [6]. Bunun yanında MapReduce'den farklı olarak MongoDB, ElephantDB, Cassandra ve HBase gibi NoSQL veri tabanları da Büyük Veri işleme ve saklamada kullanılmaktadır [2]. 2014 yılında 274 bilişim kuruluşunda gerçekleştirilen bir bilişim anketi sonucunda, bu kurumların %70'inde gerçek ya da gerçeğe yakın zamanlı Büyük Veri işleme ihtiyacı olduğu, aynı kurumların ancak %9'unun ise işlenmesi oldukça karmaşık ve teknik olarak zor olan gerçek zamanlı bu veriyi işleyebildiği belirlenmiştir [7]. MapReduce yöntemiyle

verilerin toplu (batch) olarak işlenmesi oldukça zaman almakta ve bu tekniğin gerçek zamanlı veri işlemede kullanılamayacağı bilinmektedir. Verilerin gerçek zamanlı olarak işlenerek analitik işlemlerin uygulanabilmesi, ETL (Extract Transform and Load) işlemlerinin optimize edilmesi ve görselleştirilmesi ihtiyacı ne yazık ki MapReduce yaklaşımı ile yapılamamaktadır. MapReduce algoritması, Doug Cutting ve Mike Cafarella tarafından açık kaynaklı olarak Apache Hadoop projesi ile geliştirilerek kullanıma sunulmuştur. Hadoop günlük kullanılan masa üstü bilgisayarlar üzerinde (commodity hardware) çalışabilir ve ölçeklenebilir yeteneğe sahiptir. Hadoop, Büyük Veri uygulamalarında günümüzde en yaygın kullanılan teknolojidir. Hadoop iki parçadan oluşmaktadır; bunlardan birincisi dağıtık dosya sistemi (HDFS) ve diğeri ise MapReduce programlama ve iş yönetimi yazılım çatısıdır. Hem HDFS hem de MapReduce efendi – köle (master-slave) mimariyle çalışmaktadır. Geliştirilen yazılım, bir Hadoop işi (job) olarak, iş takipçisi (jobtracker) modülüne gönderilmektedir. İş takipçisi bunu görev olarak birçok dağıtık düğümde çalışan görev takipçisine (TaskTracker) iletmektedir. Görev takipçisi aldığı görevi kendi sorumlu olduğu düğümlerde çalıştırırken görev durumu hakkında kalp atışı sinyalleri ile (Heart Beats) düzenli olarak iş takipçisine rapor etmektedir. Eğer iş herhangi bir düğüm üzerinde başarısız olursa, görevin çalıştığı düğüm çalışmaz hale gelirse ya da bir nedenden ötürü zaman aşımına uğrarsa bu durumda iş takipçisi üzerinde çalışmakta olduğu işi başka bir görev takipçisine göndererek yeniden çalıştırmaktadır. HDFS üzerinde saklanan dosyalar ile ilgili bilgiler isim düğümü (NameNode) ve ikincil isim düğümü (Secondary NameNode) üzerinde saklanmaktadır. Bir istemci, HDFS üzerinde saklanan herhangi bir dosyaya erişmek istediğinde isim düğümü üzerinden sorgulama yapar ve dosya bölütlerinin hangi düğümlerde olduğunu bulur.

Hadoop MapReduce yazılım çatısı ölçeklenebilir ve hataya karşı dirençli olacak şekilde tasarlanmıştır [8]. Ancak Giriş/Çıkış (I/O) işlemlerinin optimal ve efektif olarak çalışması için tasarlanmamıştır. Map ve Reduce işlemleri bir veri bloğu üzerinde işlem bitmeden diğer adıma geçilemeyen bir yaklaşımla dosyaya ait tüm veri blokları üzerinde çalışmaktadır. Çıkış adımında bulunan Mapper'ların işlediği veriler, Reducer'a geçmeden önce düğümlerdeki yerel disklere yazılmakta ve alfa nümerik olarak sıralanmaktadır. Bu adıma tüm Map görevleri tamamlanmadan başlanamamaktadır. Daha sonra sıralanan ve diğer Mapper'lardan gönderilen veriler birleştirilerek Reducer'lara gönderilmektedir. Reducer, Mapper'ların ürettiği bu yeni veri bloklarını kendi bulunduğu düğüme ağ üzerinden kopyalanmaktadır. Veri bloğu seviyesinde görevlerin yeniden başlaması, bire bir

gerçekleşen sıralama işlemleri ve çalışma zamanlı görev yönetimi her bir düğümün performansını düşürmektedir [2]. Ayrıca MapReduce yazılım çatısı, veri tabanı yönetim sistemlerinde olan yürütme planı (execution plan) yeteneğine sahip değildir ve düğümler arasında veri transferlerinin optimize olarak gerçekleştirilmesini de sağlayamamaktadır. Tüm bunlar bir araya geldiğinde Hadoop doğası gereği gecikme (latency) problemleri yaşamaktadır [2,16]. Bu nedenle verinin gerçek zamanlı ve toplu (batch) olarak işlenmesine uygun değildir.

Akışkan veri işleme (stream processing), verinin potansiyel değerinin güncel olarak işlenebilmesi sonucunda artması varsayımı ile ortaya çıkmıştır [9]. Bu nedenden ötürü akışkan veri işleme verilerin mümkün olduğunca çabuk olarak işlenmesi prensibine dayanmaktadır. Bu paradigmada veri işleme sistemine sürekli bir şekilde akışkan olarak ulaşmaktadır. Verinin geliş biçimi süreklilik arz ettiği için hızlı ve büyük hacimlerde gelen verinin kısıtlı hafızalar üzerinde tamamının işlenmesi mümkün olmamakta ve gelen akışkan verinin sadece belirli bir kısmı işlenebilmektedir. Akışkan verilerin işlenmesi için bazı teoriler geliştirilmiş ve teknolojik olarak gelişmeler sağlanabilmiştir [9]. Bu alanda geliştirilen ve günümüzde yaygın olarak kullanılan açık kaynaklı projeler Storm [17], S4, Samza, Kafka ve Spark Streaming'dir [10-13]. Akışkan verilerin gerçek zamanlı olarak işlenebilmesi, analizlerin çevrim içi uygulamalarda yaygın olarak saniye ya da milisaniye sürelerinde gerçekleştirilebilmesine ihtiyaç duymaktadır. Özellikle nesnelerin interneti kavramının yaygınlaşması ile sensörlerden alınan verilerin gerçek zamanlı ya da yakın gerçek zamanlı olarak işlenebilmesi gereksinimi doğmuştur. Bu türdeki akışkan verilerin işlenebilmesi için bu veri türüne özel mimarilerin geliştirilmesi zorunlu hale gelmiştir. Sensör ağlarından sürekli ve büyük hacimlerde gelen verilerin toplanması, analiz edilmesi ve saklaması işlemleri için yapılan bazı çalışmalarda, uygulamaya özel yaklaşımların sergilendiği özellikle akıllı sistemlerin akışkan veriler ile desteklenmesi için akışkan veri işleme sistemlerinin kullanıldığı görülmektedir [14,15].

Veri boyutunun aşırı miktarda büyümesi, verinin gerçek zamanlı işlenmesi için farklı modellerin geliştirilmesine neden olmuş ve bu alanda geliştirilen Apache YARN, MapReduce, Storm ve AKKA gibi Büyük Veri teknolojileri için hem veri işleme etkinliğinin hem de kaynak yönetim algoritmalarının iyileştirilmesi ihtiyacı doğmuştur [18]. Bu konuda iki farklı algoritma geliştirilmiş (JATS – Job Allocation To Supervisor, SD – Supervisor Discovery) ve yapılan denemelerde Storm'un etkinliğinin arttığı

görülmüştür [18]. Gerçek zamanlı/yakın gerçek zamanlı veri işleme teknolojilerinin mimarileri ve çalıştıkları platformlar hakkında yapılan başka bir çalışmada, gerçek zamanlı analitik işlemle için mevcut teknolojilerin her biri incelenmiştir [2]. Bu çalışmada, akışkan veri işleme ve duran veri işleme platformlarının karakteristik özellikleri ortaya konmuş ve kullanılacağı veri işleme yetenekleri belirtilmiştir. Gerçek zamanlı veri işleme platformlarının çok çeşitli olmasının yanında, hafıza kullanımının aşırı olması ve dağıtık olarak hesaplama yapılması açısından benzerlik gösterdiği belirtilmiştir. Ayrıca akışkan ve duran verilerin aynı anda işlenmesine olanak sağlayan Lamda Mimariye detaylı olarak değinilmiştir. Üç katmanlı Lamda Mimarinin akışkan ve duran veriyi işleme kabiliyeti incelenerek farklı teknolojilerin birlikte kullanılabilirliği ve esnekliği konusunda avantaj sağladığı tespit edilmiştir.

2010 yılında yapılan bir çalışmada [19], tam anlamıyla bir Lamda Mimari olmasa da oldukça benzer özelliklere sahip bir mimari tasarım sunulmuştur. MapReduce ve akışkan veri işleme yeteneklerinin bir araya getirilmesi fikri ile yapısında IBM teknolojilerinin kullanıldığı DEDUCE yazılım çatısı geliştirilmiş ve finansal market veri işleme alanında denemeler yapılmıştır. Söz konusu sistemde, farklı kaynaklardan gelen yeni finans verileri ve kayıt altına alınmış diğer finans verileri, MapReduce ile toplu olarak analiz eden bir model geliştirilmiştir. Buradan elde edilen sonuçlar ve model yardımıyla anlık olarak gelen akışkan veriler normalize edilerek bir veri tabanında saklanmaktadır. İlk bakışta benzerlik gösterse de, söz konusu mimari kesinlikle Lamda Mimari olarak sınıflandırılamamaktadır. Lamda Mimarinin duran ve akışkan verileri aynı anda sorgulaya bilme özelliği DEDUCE sisteminde bulunmamaktadır. Bu yapıda akışkan veriler, duran verilerin analizi sonucu elde edilen model ile birleştirilmekte ve elde edilen sonuçların bir veri tabanında saklanmasına olanak sağlamaktadır. Ancak yine de duran veri ile hareketli verinin bir arada kullanıldığını gösteren bir fikir olarak dikkat çekmektedir.

Günümüzde, Büyük Veri alanında operasyonel modeller ve veri merkezli mimarilere yönelim görülmektedir [20]. Bu maksatla geliştirilen BDAF – Big Data Architecture; yazılım çatısı, Büyük Veri altyapısı, Büyük Veri analitikleri, veri yapıları ve modelleri, Büyük Veri yaşam döngüsü yönetimi ve güvenlik konularında karşılaşılan güçlükleri işaret ederek, bu konularda çözüm üretmek için geliştirilmiştir. Önceleri Büyük Veri için kullanılan 3V tanımı, bu çalışmada 5V olarak ele alınmıştır. “All-Data Infrastructure” yapısı içerisinde Büyük Veri teknolojilerinin veri merkezli olarak tahsis edilmesini ve

büyük verilerin işlenmesi için genel mimarilerin ortaya konmasını sağlamaktadır. Bu mimariler dikkatle incelendiğinde tüm ihtiyaçların karşılandığı bir ekosistem kurulumu öngörülse de, özünde Lamda Mimariye işaret eden yaklaşımlarla karşılaşılmaktadır.

Akışkan ve duran verilerin birlikte işlenmesi için mimari geliştirme ihtiyacı farklı alanlarda kendini göstermeye başlamıştır. Bu alanlardan birisi de uzaktan algılama ve görüntü işleme sistemleridir. Uydulardan elde edilen görüntülerin gerçek zamanlı ve toplu olarak işlenmesi üzerine yapılan çalışmalarda [21], farklı yaklaşımlar sergilenmektedir. Tam olarak Lamda Mimari olmasa da, büyük, duran ve akışkan uydu görüntülerinin birlikte işlendiği bu çalışmada; uydulardan elde edilen görüntülerin ön işleme gereksinimleri akışkan veri işleme altyapısı ile gerçekleştirildikten sonra Büyük Veri altyapısı üzerinde saklanmıştır. Daha sonra MapReduce [22] yardımıyla geliştirilen algoritmalar ile su, deniz ve buz gibi hedefler bulunabilmektedir. Bu yaklaşım ve altyapı ile oldukça başarılı sonuçlar elde edilmiştir. Veri işleme süresi ve saklama kabiliyeti geliştirilerek akışkan verilerin kayba uğramadan anlık olarak işlenmesi ve durağan hale geldikten sonra yapılan analizlerle hedeflerin bulunması, veri merkezli bir işleme ihtiyacını ortaya koymaktadır. Farklı nitelikteki veri türlerinin birlikte işlenmesi kabiliyetinin kazanılmasının mevcut teknolojiler ile mümkün olduğu görülmektedir.

Büyük Veri sistemlerinin tasarım prensipleri araştırıldığında [23], öncelikle analitik algoritmaların geniş ölçekte çalışabilecek şekilde paralelleştirilmesine ve istenildiğinde doğrusal olarak ölçeklenebilecek bir yapıda olmasına ihtiyaç duyulmaktadır. Yeni teknolojiler geliştirilmesinin yanında, yeni düşünce ve yaklaşımların da geliştirilmesi gerekmektedir. Bu kapsamda bir Büyük Veri sisteminin geliştirilmesi ve tasarlanabilmesi için yedi kaideden faydalanılmaktadır [23]:

- Yetkin mimariler ve yazılım çatıları: Büyük verilerin işlenmesi için veri türüne uygun bir mimari ve yazılım çatısı seçilmesi gerekmektedir. Geleneksel bilgi sistemlerinde işlenecek veriler genellikle veri tabanı yönetim sistemleri içerisinde ve veri ambarlarında tutulmaktadırlar. Burada farklı analitik teknikler ile işlenerek saklanmaktadırlar. Büyük Veri işleme sistemleri geleneksel mimarilere göre daha yüksek düzeyli mimarilere ihtiyaç duymaktadır. Akışkan ve durağan verilerin işlenmesi için farklı gereksinimler bulunmaktadır. Akışkan verilerin işlenmesinde NoSQL tabanlı anahtar-değer veri-saklama yapıları yüksek performanslı ve dizin (index) tabanlı

çağırma işlemlerine olanak sağlamaktadır. Farklı türdeki veri – duyarlı (data intensive) uygulamalar için, farklı türde ve uygun mimari tasarımlara gidilmelidir [23]. Örnek olarak Lamda Mimari [2, 3, 23], gerçek zamanlı verinin işlenebilmesi için üç farklı katman tanımlamaktadır. Bu katmanlar durağan veri katmanı, akışkan veri katmanı ve sunum katmanı olarak tanımlanır ancak bu mimari de her Büyük Veri uygulaması için uygun değildir [23].

- Çeşitli analitik metotların desteklenmesi: Büyük Veri uygulamaları genellikle bir ya da birden çok analitik metod ve disiplinlerin çözebileceğinden daha karmaşık işlemler gerektirmektedir. Veri bilimi, geniş bir yelpazede çalışılması gereken konulara ve yaklaşımlara gerek duymaktadır. Veri madenciliği, istatistiksel analiz, makine öğrenimi, dağıtık programlama ve görselleştirme yöntemleri, gerçek zamanlı ve hafıza üzerinde (in-memory) analizler ve insan makine etkileşimi yetenekleri kullanılmaktadır. Bu metotların ve yöntemlerin mutlaka farklı Büyük Veri platformlarında bulunması gerekmektedir.
- Veri Hacmi: Büyük Veri işlemek için kullanılan her bir aracın kendine özgü kısıtları bulunmaktadır. Daha önce de belirtildiği gibi uygun aracın uygun işlem için kullanılması gerekmektedir. Bilginin yüksek hızda artmasıyla başa çıkabilecek şekilde altyapıların oluşturulabilmesi yeteneği mutlaka mevcut olmalıdır.
- Analiz işleminin verinin olduğu yerde yapılması: Büyük Veri setleri aşırı şekilde yüksek hacimli olduğu için söz konusu verilerin bir araya getirilerek bir ya da birden fazla merkezde toplanması mümkün olmamaktadır. Aksine, analiz işlemlerinin verinin bulunduğu yerlerde yapılması daha doğru bir yaklaşım olacaktır. Platform ve mimari tasarımlarda bu konunun dikkate alınması gerekmektedir.
- Hesaplamaların ve veri işleminin hafıza üzerinde dağıtık çalışabilmesi: Farklı konumlarda bulunan verilerin işlenmesi için dağıtık veri işleme yeteneğinin olması gerekmektedir. Hafıza üzerinde çalışabilen analitiklerin popüler olmasının nedeni sabit sürücü gibi yavaş ortamlar üzerinde çalışmaması, aksine, hafıza üzerinde çalışabildiği için işlemleri hızlı olarak tamamlayabilmesidir. Hafıza tabanlı analitikler gerçek zamanlı verilerin işlenebilmesi için gereklidir [24].
- Hafıza tabanlı depolama için veri depolamanın dağıtık olması: Verilerin parçalanmış ve dağıtık olarak saklanabilmesi işlemlerin veri merkezlerinde dağıtık olarak gerçekleştirilebilmesine olanak sağlamaktadır. Veriler bir bulutta saklandığında

kullanıcılar yüksek kapasiteli işlem yeteneğine sahip donanımlar üzerinde büyük verilerini işleyebilmektedirler.

- İşleme ve veri birimleri arasında koordinasyona ihtiyaç duyulması: Etkinliğin, ölçeklenebilirliğin ve hataya toleranslı Büyük Veri sistemlerinin iyileştirilebilmesi için veri işleme birimleri ile veri birimleri arasında koordinasyonun sağlanması gerekmektedir. Bu nedenle Storm ve S4 gibi gerçek zamanlı veri işleyebilen projeler ZooKeeper gibi kümeleme koordinasyon yazılım çatılarına ihtiyaç duymaktadırlar. Bu kaide düşük gecikmeli çalışması gereken gerçek zamanlı sistemler için gereklidir.

Büyük Veri mimarisinin kullanımı ile ilgili çalışmalara çeşitli yayınlarda yer verilmektedir [25]. Özellikle Facebook, Twitter ve LinkedIn gibi şirketlerin sosyal medya uygulamaları farklı ve etkin teknolojilerin kullanımını gerektirmektedir. Facebook yapısal ve akışkan türdeki verileri kullanıcılarından toplamaktadır. Bunların bir araya getirdikten sonra toplu olarak analizini gerçekleştirmektedir. LinkedIn de yapısal ve akışkan türdeki verilerin kullanıcılardan toplanması ile elde ettiği verileri toplu olarak işleyerek veri bilimcilerinin kullanımına sunmaktadır. Gerçekleştirilen analizler ile öneri motoru gibi uygulamalar sağlanabilmektedir. Twitter ise kendi sistemi üzerinde üretilen gerçek zamanlı verileri işlemek zorunda kalmaktadır. Bu nedenle kullandığı mimari daha farklı bir yapıdadır. Toplanan verilerin işlenmesi ile kullanıcıların başka kullanıcıları takip edebilmesi için önerilerde bulunmaktadır. Netflix şirketi ise, kendi sitesinde yayınladığı çevrim içi filmleri kullanıcıların ziyareti ve izlemesi sırasında topladığı verilerin bir araya getirilmesi ve analiz edilmesi ile yeni film önerilerinde bulunmaktadır. Burada anlatılan ve incelenen mimarilerde görüldüğü üzere birçok şirket farkında olmadan Lamda Mimariyi kullanmakta ya da buna benzer bir mimari üzerinde çalışmaktadır.

Avrupa birliği [26] politika ve teknoloji raporunda, mevcut yazılım çatılarının sıklıkla bir araya getirilerek; karmaşık analizlerde toplu veri işleme yöntemlerinin, çevrim içi analizlerde gerçek zamanlı sorgulama yöntemlerinin ve sürekli sorgulama ile akışkan verilerin incelendiğini belirtmiştir. Bu şekilde farklı teknik yeteneklerin bir araya getirilmesi ile depolama ve işleme için ölçeklenebilir, düşük gecikmeli bir yapı oluşturulmakta ve karmaşık veri işleme yapılabilmektedir. Bu yapı için üç katmanlı Lamda Mimari örnek olarak verilmiştir.

Nesnelerin interneti teknolojileri (IoT) için Büyük Veri teknolojilerinin kullanılması kaçınılmazdır [27]. Gerçekleştirilen bu çalışmada, akıllı ev konsepti için geliştirilen bir sistemde farklı sensörlerden gelen verilerin gerçek zamanlı ve çevrim dışı analizi için bir Lamda Mimari kullanılmıştır. Geliştirilen üç katmanlı mimaride akışkan verilerin işlenmesi için Storm ve durağan verilerin işlenmesinde NoSQL tabanlı MongoDB kullanılmaktadır. Farklı kaynaklardan gelen veriler gerçek zamanlı ve çevrim dışı olarak analiz edilmektedir. Sunum katmanı olarak yine MongoDB kullanılması tam olarak orijinal Lamda Mimariye (LM) [1] uymasa da, tasarım, veri işleme yeteneği açısından Lamda Mimariye uygun olarak geliştirilmiştir.

Lamda Mimari akışkan veri işleme uygulamalarının MapReduce ve Storm ya da benzer sistemlerin üzerine kurulduğu bir mimaridir [28]. Bu kavram yeni gelişmekte olsa da sürpriz şekilde popülerlik kazanmıştır. Mimari açısından farklı şekillerde tanımlanabilmekte, spesifik bir teknoloji kullanılma zorunluluğu bulunmamaktadır. Lamda Mimarın sorunlu olarak görülen özelliklerinden ötürü eleştiriler de almaktadır [28]. Karmaşık ve dağınık bir sistemde geliştirilen kodların idame ettirilmesi oldukça zor bir iştir. Hadoop ve Storm gibi teknolojilerde dağınık çalışacak uygulamalar geliştirilmesi karmaşık olarak görülmektedir. Geliştirilecek kodlar, kullandığı teknolojiye özgü olacaktır. Durağan ve akışkan veriler için farklı teknolojiler kullanılacağı için, farklı kodlar geliştirilecek ve aynı mimari içerisinde farklı sistemler üzerinde çalışacaktır. Bu sorunun aşılması için Summingbird gibi yazılım çatıları geliştirilmiştir. Bu yazılım çatısında, kod daha özet (abstract) düzeyli bir programlama dilinde geliştirilecek iki farklı teknoloji için kullanılması sağlanmaktadır. İki defa kodlamadan bu şekilde kaçınılsa da iki farklı sistemde kodların test edilmesi ve hatalarının ayıklanması gerekmektedir. Bu yaklaşım dolayısı ile Hadoop ekosistemindeki diğer başarılı yazılım çatıları gerektiğinde kullanılamamaktadır.

Sorun olarak görülen bu durum, bu tez çalışmasında geliştirilen bir yöntemle ortadan kaldırılmıştır. Duran ve akışkan verilerin çok hızlı ve yinelemeli(recursive) algoritmaları da destekleyecek şekilde işlenmesi için bir defa kod geliştirip iki katmanda kullanılması bu çalışmanın ana hedeflerinden birisidir.

Birden fazla yazılım geliştirme ve çalışır hale getirme süreçleri LM için engel olarak görülse de gerçekte çok önemli bir sorun değildir. Aslında bu şekilde yazılım geliştirilirken

farklı teknolojiler üzerinde farklı kodlamaların yapılabilmesi ile iki farklı veri türüyle uğraşan katmanlarda ayrı algoritma kullanılabilmesi sonucu esneklik sağlanabilmektedir. İki farklı katmanda kullanılan teknolojilerden birisinin yeteneği olduğu bir alanda, diğeri bu yeteneğe sahip olmayabilmektedir. Dolayısı katmanların veri işleme için birbirlerinin eksik yönlerini tamamlaması sağlanmaktadır.

LM yerine önerilen Kappa Mimari ise bu sorunu ortadan kaldırmakta olsa da, henüz tam olgunlaşmamış ve tek bir teknolojinin üzerine inşa edilmiş bir mimaridir. Mimari üzerinde gerçekleştirilecek sorgulamalar sürekli bir şekilde verilerin işlenmesine ve aşırı donanımlar üzerinde aşırı bir yük oluşmasına neden olmaktadır. Bu konuda başka bir eleştiri ise sunum katmanında NoSQL tabanlı veri tabanları kullanıldığı için SQL kullanmaya alışkın kullanıcılar tarafından etkin olarak geliştirme yapılamamasıdır. Aslında bu da çok önemli bir sorun olarak değerlendirilmemektedir. NoSQL veri tabanları SQL veri tabanlarından çok daha üstün yetenek ve veri işleme kabiliyetlerine sahiptir. Birçok NoSQL veri tabanında zaten doğal olarak SQL dili kullanılarak analiz desteği gelmektedir. Tasarım yapılırken teknik olarak bu konu dikkate alınırsa, Lamda Mimarinin bir birinden farklı teknolojiler kullanılarak oluşturulabilme ve esnek bileşen seçilebilme özelliği ile SQL destekleyen bir sunum katmanının bulundurulması sağlanabilecektir.

ADS-B mesajlarının işlenmesi için geliştirilen bir Lamda Mimaride (Demirezen, 2015), iki farklı Büyük Veri teknolojisi kullanılarak bir mimari geliştirilmiştir. Bu çalışmada akışkan veri katmanında Apache Storm, duran veri ve sunum katmanında MongoDB teknolojileri kullanılmıştır. Burada temel amaç duran veri katmanı ve sunum katmanı için aynı teknolojinin kullanımının sağlanmasıyla geliştirme yükünün azaltılmasıdır. Gerçek zamanlı veriler AVK ile işlenerek sunum katmanındaki MongoDB'ye iletilmektedir. MongoDB; MapReduce desteğine sahip olması ve yüksek hızda gerçek zamanlı veri ithalini oldukça iyi bir şekilde desteklemesinden ötürü hem sunum katmanında ve hem de duran veri katmanında kullanılmıştır. Bu çalışmada Lamda Mimarinin tüm özellikleri net olarak uygulanmıştır. Duran veri katmanının, akışkan veri katmanından gelen verileri düzeltici etkisinin kullanıldığı görülmektedir. Ancak iki farklı katman için geliştirilmede sağlanan iyileştirilme, bu çalışmada sunum katmanı ve duran veri katmanının birleştirilmesi ile sağlanmış ve oldukça iyi sonuçlar elde edilmiştir.

İki farklı akışkan veri işleme teknolojisinin bir araya getirilmesi ile Lamda Mimari oluşturulduğu bir çalışmada [30], duran veri kontrollü bir şekilde akışkan hale getirilerek veri işleme gerçekleştirilmiştir. Halka açık (public) veriler üzerinden yürütülen bu çalışmada duran veriler geliştirilen bir yazılımla Kafka üzerinden sisteme gönderilmiş, aynı anda hem Storm hem de Samza teknolojilerinin kullanılmasıyla bu veriler işlenerek sunum katmanına iletilmiştir. Lamda Mimari geliştirildiği belirtilse de yapılan bu çalışmanın bir Lamda Mimari olmadığı değerlendirilmektedir. Özellikle sunum katmanından yok denecek kadar az bahsedilmesi, duran veri işleme yapılmaması, duran veri katmanının akışkan veri katmanının ürettiği sonuçları düzeltmede kullanılması gibi konuların eksik olduğu görülmektedir. Ancak bu konuda gerçekleştirilmiş oldukça detaylı bir çalışmadır. Tasarlanan mimarinin test edilmesi için kullanılan yöntem hatalar içermekte olup geliştirilmeye ve tartışmalı noktaların giderilmesine ihtiyaç duymaktadır.

Çok ajanlı sistem (multi-agent system) paradigması kullanılarak gerçek zamanlı Büyük Veri işleme platformu ile bir öneri motoru (recommendation engine) uygulamasında lamda mimari kullanılmıştır [32]. Büyük Veri işlemek için kullanılan bir Lamda Mimari heterojen çok ajanlı çevre olarak modellenenmektedir. Üç farklı katman karakteristikleri açısından farklılık göstermekte ancak bir birleri ile etkileşimde bulunarak çalışmaktadır. Etkileşim için kullanılan haberleşme yöntemleri çok ajanlı sistem yaklaşımı kullanılarak basitleştirilebilir. Otonom ve dağıtık olarak bulunan bu ajanlar, veri işleme, bir araya getirme, veri alma gibi spesifik bir işlem için kullanılabilir. Ajanlar arasında koordinasyon mesaj geçirme (message passing) ile yapılabilmektedir. Büyük ajanlar aynı yöntemi kullanacağı için birbirleri ile olan entegrasyon basitleştirilmiş olmaktadır. Teorik olarak gerçekleştirilen bu çalışmanın pratikte kullanımı ve doğrulaması mevcut değildir.

Metin tabanlı verilerin analizi için Lamda Mimariden faydalanılmıştır [33]. Amazon Web Servisleri kullanılarak geliştirilen örnek bir uygulamada sadece duran veri katmanında veriler işlenmiştir. Ancak bu çalışmada akışkan veri katmanı ve sunum katmanı kullanılmadığı için gerekli Lamda Mimari yeteneklerini yapısında barındırmamaktadır.

Büyük Veri sisteminin düşük zaman gecikmeli sorgulara yanıt verilecek şekilde tasarlanabilmesi ihtiyacı Lamda Mimarinin ortaya çıkmasına yol açmıştır [34]. Gerçek zamanlı sonuçların elde edilmesi için duran ve akışkan verilerin işlenmesi birleştirilmektedir. Analiz sonuçlarının sürekli olarak sorgulanmasına gerek olmadığı

durumlarda her iki katmanında kullanılmasına gerek yoktur. Talep edildiği durumda çalışacak ve talep ortadan kalktığında çalışmasına gerek olmayan akışkan veri işleme katmanı tasarımı ile kaynakların daha efektif kullanımı mümkün olmaktadır. Bu nedenle yapılan çalışmalarda duran veri katmanının çalışma sürelerinin önceden kestirilmesi ve bu sürelerle bakılarak verilen kararlarla akışkan veri katmanının kullanılması farklı bir yaklaşım olmaktadır. Performans modeli kullanılarak bir karar verici model kullanımı geliştirilmiştir. Ancak sunum katmanının tükettiği kaynaklar performans modeline dâhil edilmemiştir. Bir rüzgâr tribünü verisi üzerinde test edilen model ile sadece duran veriler işlenebilmektedir.

Literatür taramasında fark edildiği üzere, Büyük Veri mimarileri ve gerektirdiği yetenekler hakkında çok fazla sayıda yayına rastlansa da, Lamda Mimari Büyük Veri sistemleri için önemli bir yayına rastlanamamıştır [35]. Bulunan yayınlar genellikle akademik olmayan, mühendislerin teknik olarak tecrübelerini ya da görüşlerini paylaştığı yazılar olarak karşımıza çıkmaktadır. Akademik olarak yayınlanan çalışmaların Lamda Mimari kaidelerini tam olarak yerine getirmediği, aynı anda iki verinin işlenip farklı amaçlarda kullandığı, sunum katmanının neredeyse hiç dikkate alınmadığı belirlenmiştir.

Bu çalışmada, yenilikçi bir yaklaşım ile Lamda Mimari konseptinin her bir bileşeninin tam olarak kullanıldığı, ileri sürülen eleştiri ve mevcut handikapları ortadan kaldıracak özgün bir mimari ve konsept oluşturulmuş ve gerçek verilerle farklı koşullar altında test edilmiştir.

2.2. Duran Veri İşleme Modeli

Duran veri analizi, verinin boyutu üzerine odaklanmış bir veri işleme modelidir [36]. Duran verilerin toplu olarak işlenebilmesi 2003 yılında Google'ın geliştirdiği dosya sistemi, (GFS) [37] ve MapReduce yazılım [6] çatısı ile mümkün hale gelmiştir. Ancak Hadoop; açık kaynak kodlu olarak geliştirilmesi [16] ve olgunlaşmasıyla birlikte bu alanda en yaygın kullanılan teknoloji olarak karşımıza çıkmaktadır.

Duran veri işleme, büyük miktardaki statik verinin işlenmesi için geliştirilmiş bir çözümdür. Bu veri türü işleneceği zaman zaten sistemde depolanmış halde bulunmaktadır. Veri işlenmeye başlandığında yeni gelen veriler dikkate alınmayarak üzerinde herhangi bir

işlem yapılmamakta sadece başlama anına kadar olan veriler üzerinde çalışılmaktadır. Duran veri işlemenin en temel özelliği ölçeklenebilir olmasıdır. Ölçeklenebilirliğin sağlanması için MapReduce gibi paralel dağıtık veri işleme yazılım çatıları kullanılmaktadır. MapReduce zaman içerisinde duran veri işlemede standart hale gelmiştir. MapReduce veri işleme modelinin kullanılması ile çeşitli avantajlara sahip olduğu avantajlar şöyle sıralanmaktadır;

- Verinin basit ve birleştirilmiş bir şekilde kullanılması
- Ölçeklenebilir olması
- Farklı donanımların alt yapıda kullanılması, donanımların bir biri arasındaki ağ haberleşmesi, donanım arızaları gibi uğraşılması zor konularda dağıtık programlama işlemlerini ve karmaşıklığı basitleştirmesi ve programcıdan izole etmesidir.

Bununla birlikte MapReduce bazı sorunları ve kısıtlamaları da beraberinde getirmektedir. Birçok analiz yöntemi ve algoritması arttırımsal ya da yinelemeli olarak çalışmaktadır. Bu tip işlemlerin gerçekleştirilmesi MapReduce ile oldukça zorlaşmaktadır. Duran veri işleme oldukça güvenilir olmasına karşın tamamlanması çok uzun zaman alabilmektedir. Bu nedenle düşük gecikme gereken uygulamalarda kullanılamamaktadır. Duran veri işleme başlatıldığında yarıda kesilememekte ya da bu sırada gelen yeni verilerinde işlenmesi için yeniden ayarlanamamaktadır.

Bir MapReduce uygulaması Map ve Reduce olarak iki fazdan oluşmaktadır. Map fazının girişinde anahtar-değer (key-value) ikilisi kullanılmaktadır. Map fonksiyonu bu anahtar-değer ikilisini aldıktan ve gerekli işlemleri uyguladıktan sonra ara değer olarak yine anahtar-değer ikilileri yayınlamaktadır. Reduce fonksiyonu ise ara değer olarak yayınlanan anahtar-değer ikililerinden her bir anahtar için oluşturulmuş bir listeyi giriş olarak almakta ve bunların üzerinde kendi işlemlerini yaptıktan sonra son/nihai, anahtar-değer ikililerini yayınlamaktadır.

MapReduce dağıtık Büyük Veri işleme veri modeli olarak yaygın olarak kullanılsa da, bu alanda kullanılmakta olan başka veri işleme modelleri de bulunmaktadır. Toplu – Senkron – Paralel veri işleme modeli (Bulk Synchronous Parallel - BSP), MapReduce'den önce de var olmasına karşın yakın zamanda tekrar popülerlik kazanmıştır. Bu modelde program bir dizi süper adımlar (supersteps) olarak tanımlanmaktadır. Her bir süper adım, daha küçük ve yerel olarak hesaplama yapan adımlardan oluşmaktadır. Süper adımlar çalıştırıldıktan

sonra bir senkronizasyon bariyerine ulaşmaktadır. Bu senkronizasyon noktasında sistem bütün işlemci birimlerin ağ haberleşmesi ve hesaplamalarını bitirmesini beklemektedir. Bu programlama modelinin graf tabanlı hesaplama algoritmalarında daha etkin olduğu bilinmektedir.

Daha öncede belirtildiği gibi bu tür veri işleme modellerinin interaktif ya da akışkan veriler üzerinde kullanımı uygun değildir. Bu nedenle daha hızlı çalışan veri analizi modelleri geliştirilmiştir. Bu analiz modeli ile veri oluştuktan hemen sonra canlı olarak işlenebilmektedir. Verinin aktarım halindeyken uygulanan analizlerin sonuçları hızlı bir şekilde elde edilerek kullanılmaktadır.

2.3. Akışkan Veri İşleme Modeli

Akışkan veri işleme verinin hızına ve gecikmesine odaklanmış bir veri işleme modelidir [37]. Bu veri işleme yöntemi duran veri işleme yöntemlerine verilerin paralel ve dağıtık olarak işlenmesi yönünden benzemektedir. Düşük gecikme değerlerini elde edebilmek için, hafızada küçük setler halinde birikmiş olan veriler hızlı ve sürekli bir şekilde defalarca işlenmektedir. Bu açıdan düşünüldüğünde, depolanmış verinin sabit sürücüler yerine hafızada bulunduğu, ufak sayıdaki duran verilerin sonsuz defa işlenmesi olarak düşünülebilmektedir. Başka bir deyişle bu yöntem veri işlerken sabit disk kullanmamaktadır.

Akıllı şehir, felaket yönetimi, lojistik, sosyal medya, mobil cihazlar, nesnelerin interneti (IoT) gibi bazı uygulamalar farklı kaynaklardan gelen akışkan verilerin gerçek zamanlı olarak işlenmesine ihtiyaç duymaktadır.

2.4. Hibrit Veri İşleme Modeli

Birçok uygulama alanı akışkan veri işleme ve duran veri işleme modellerinin birlikte kullanımına ihtiyaç duymaktadır. Bu veri modeli her iki veri modelinin birlikte kullanıldığı hibrit bir yapı ile oluşturulabilmektedir. Bu tip bir veri modeline Lamda Mimari adı verilmektedir [1]. Lamda Mimari, verilerin dağıtık bir dosya sisteminde saklandığı ve ana veri seti üzerinde çalışan duran veri işleme katmanını, duran veri katmanında oluşturulan

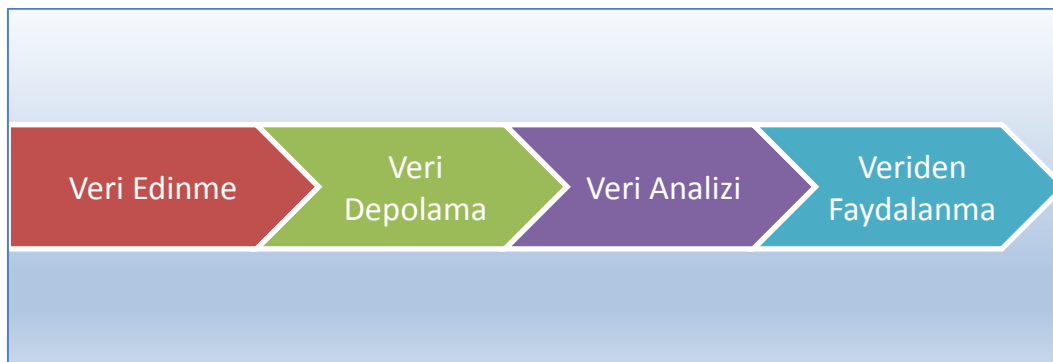
görüntülerin sorgulanabilir hale getirildiği sunum katmanını ve yeni gelen verilerin düşük gecikmeli olarak işlendiği akışkan veri katmanını yapısında barındırmaktadır [37].

İşlenen verilerden sonuçların tam olarak elde edilebilmesi için, duran veri katmanı ve akışkan veri katmanında oluşan sorgu odaklı görüntülerin ayrı ayrı sorgulanıp sonuçların birleştirilmesi gerekmektedir.

Yeni gelen veriler duran ve akışkan veri katmanlarının her ikisine de iletilmektedir. Duran veri katmanı bir döngü içerisinde tüm veri setini işlemektedir. Ancak bu toplu veri işleme çok uzun zaman almaktadır. Bu süre içerisinde yeni gelen verilerin işlenmesi duran veri katmanı tarafından yapılmamaktadır. Bu gecikmenin giderilmesi için duran veri katmanı tarafından işlenmeyen, sadece yeni gelen veriler akışkan veri katmanı tarafından işlenmektedir. Her iki katman da veri işleme sonuçlarını bir veri tabanına ileterek sorgulamaya hazır hale getirmektedir.

2.5. Büyük Veri İşleme Teknolojileri

Genellikle veri işleme çalışmalarında veri elde etme, geçici ya da kalıcı olarak veri saklama, veri analizi ve sonuçların üretilmesi adımları takip edilmektedir [37]. Şekil 2.1’de Büyük Veri işleme yaşam döngüsü gösterilmiştir.



Şekil 2.1. Büyük Veri işleme yaşam döngüsü

Büyük Veri işleme yaşam döngüsü dört fazdan oluşmaktadır. Bu modelin katı bir şekilde uygulanması gerekmemektedir. Bazı fazların gerçekleştirilebilmesi için bir önceki fazın da gerçekleştirilmiş olması gerekmektedir. Örnek olarak veri depolama fazından önce mutlaka veri edinme fazının yürütülmesi gerekmektedir. Akışkan veri işleme modelinde

gelen verilerin sürekli olarak depolanıp daha sonra işlenmesinin mümkün olduğu gibi, geldiği zaman dilimi içerisinde hızlı bir şekilde işlenip sonuçların elde edilmesi de mümkündür. Büyük Veri işleme yaşam döngüsü adımları aşağıdaki gibi açıklanabilir [37];

- Veri edinme: Bu fazda verilerin farklı kaynaklardan temin edilmesi işlemleri gerçekleştirilmektedir. Tüm veri işleme yaşam döngüsünün ilk fazıdır. Veriler duran veri kaynaklarından ya da gerçek zamanlı işlenmesi için akışkan veri kaynaklarından elde edilebilmektedir.
- Veri depolama: Bu faz elde edilen büyük verilerin depolanması ile ilgilenmektedir. Elde edilen verinin işlenebilmesi için hafızada ya da bellekle saklanabiliyor olması gerekmektedir.
- Veri analizi: Bu fazda verilerin işlenebilmesi ve analizi için, farklı uygulamalarda birçok model, teknik, yöntem ve algoritma kullanılabilmektedir. Gerçekleştirilecek analizlere göre uygulanan teknikler değişiklik göstermektedir.
- Veriden faydalanma(sonuçlar): Bu fazda verilerin işlenmesi sonucu elde edilen sonuçlar kullanılmaktadır. Başka bir deyişle bir önceki fazda üretilen gözlemler ve sonuçlar rapor haline getirilmektedir. Uygulama alanına göre verilerden faydalanma biçimi de farklılık gösterebilecektir.

Büyük Veri sistemleri dağıtık mimarilerde çalışmaktadır. Birçoğu parçala ve işle yaklaşımı ile verileri işlenebilecek daha ufak parçalara ayırmakta ve bu parçaları işledikten sonra sonuçları bir araya getirmektedir. Büyük Veri sistemlerinde verilerin saklanması için donanımlar üzerinde oluşabilecek hatalara karşı bir önlem alınması gerekmektedir. Bu önlem işlenecek verinin daha ufak parçalara ayrıldığında birden fazla kopyasının birden fazla donanım üzerinde saklanması ile alınmaktadır. Büyük bir dosya işleneceği zaman ya da analiz sonucunda oluşturulduğunda, birden fazla sayıda ufak parçalara bölünerek depolama alanlarında kopyaları saklanmaktadır. Eğer analiz sırasında hata oluşursa, diğer kopyalar üzerinde analize devam edilmektedir. Bir donanımın arıza yapması durumunda sistem etkilenmeden kopya veriler üzerinde çalışmaya devam edebilmektedir. Büyük Veri sistemleri geleneksel sistemlerden bu yönde farklılık göstermektedir. Dağıtık sistemlerin bu farklı özellikleri CAP teoreminde açıklanmaktadır [32].

Veri türlerine ve uygulama alanına göre Büyük Veri işleme sistemlerinde farklı teknolojiler kullanılmaktadır. Pratikte birçok Büyük Veri uygulaması geleneksel ACID (Atomicity, Consistency, Isolation, Durability) özelliklerini taşımamaktadır. CAP teoreminin belirttiği gibi [32, 38-39] dağıtık çalışan bir Büyük Veri işleme sisteminde aynı anda aşağıda belirtilen özelliklerin en fazla iki tanesi sağlanabilmektedir:

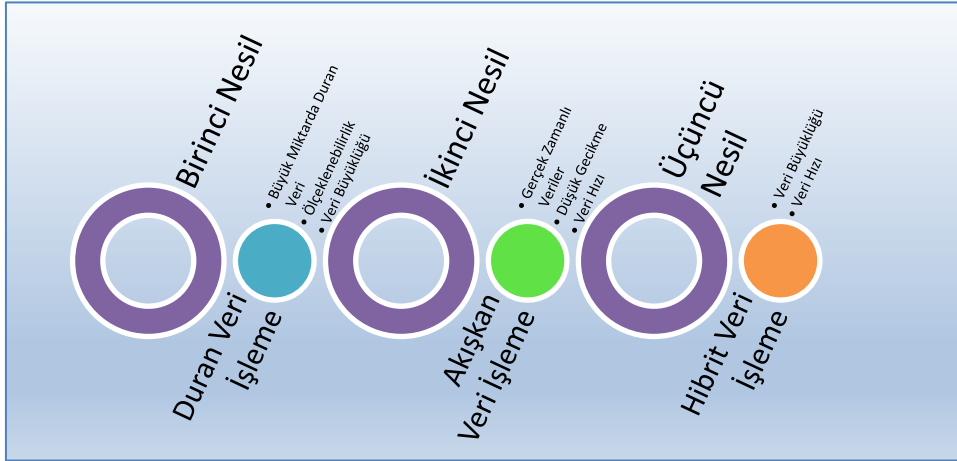
- Tutarlılık(Consistency): Bütün düğümlerin aynı anda aynı veriyi görebilmesi,
- Ulaşılabilirlik(Availability): Sistemde var olan bütün düğümlere bir istek yapıldığında hepsinden yanıt dönebilme garantisi,
- Bölüt Toleransı(Partition Tolerans): Sistemin bir parçasının çalışmaz hale geldiğinde ya da bir miktar veri kaybolduğunda, sistemin çalışmaya devam edebilmesi.

CAP teoreminde anlatılmak istenen bir veri sisteminde tüm donanımlar ağ üzerinde haberleşemediği durumda ne olabileceğidir. Bir sistemin tutarlı ve ulaşılabilir olması, fakat bölüt toleranslı olmamasının bir anlamı yoktur. Teoremin anlatmak istediği husus verinin bölütlere ayrıldığı durumdaki veri sisteminin durumudur. Bir dağıtık veri sistemi depolanan veri bölütlere ayrıldığında tutarlı ya da ulaşılabilir olarak çalışabilmekte, ancak her iki özelliğe birden sahip olamamaktadır. Eğer tutarlılık özelliği seçilirse bazen sorgu sonuçları cevap yerine hata bilgisi döndürebilecektir. Eğer bu sistemlerde ulaşılabilirlik özelliği seçilirse bölütlerin ağ üzerindeki farklı düğümlere transfer edilmeleri sırasında sorgulama yapıldığında eksik sonuçlar ile karşılaşılacaktır. Ulaşılabilir bir sistemin tutarlı olabilmesi için en iyi çalışma şekli, sistemin ağ üzerinden bölütleri transfer etmesi tamamlandıktan sonra tutarlı hale gelmesidir. Verinin dağıtık bir şekilde Büyük Veri sisteminde en ham halde tutulması ve gerektiğinde bu veriden yeniden hesaplamaların yapılması sayesinde CAP teoremi izole edilebilmektedir. Bu şekilde çalışan sistemler eninde sonunda güvenilir (eventually accurate) sistemlerdir. Lamda Mimari de bu özelliği bünyesinde barındırmaktadır.

Büyük Veri işleme teknolojileri geçmişten günümüze kadar incelendiğinde üç farklı nesil(generation) belirlenmektedir [37]. Şekil 2.2’de gösterildiği gibi, duran veri işleme 2003 yılında Google’ın Google file sistemi ve MapReduce çatısını geliştirmesi ve bununla ilgili makalelerini yayınlaması ile başladığı kabul edilebilir. Bu yıllarda iş dünyası ve kurumların gerçekten de Büyük Veri ile karşı karşıya geldiği söylenememektedir. Bu nedenle gerçekte Büyük Veri işleme Hadoop teknolojisinin geliştirilmesi ile 2006 yılında

başlamıştır. Hadoop yaygın olarak kullanılan en olgunlaşmış duran veri işleme teknolojisidir. Günümüzde Hadoop teknolojisinden daha sonra Spark'ın geliştirilmesi ile başka bir duran veri işleme teknolojisi kullanılmaya başlanmıştır. Spark, Hadoop'un veri işleme alanındaki birçok kısıtlarını ortadan kaldırarak günümüzdeki en önemli Büyük Veri işleme teknolojisi haline gelmiştir.

Büyük Veri işleme teknolojilerinde ikinci nesil, gerçek zamanlı olarak gelen verilerin işlenmesi için teknolojilerin geliştirilmesi ile başlamıştır [37]. Yahoo!, Twitter gibi şirketlerin sadece duran verileri değil gerçek zamanlı olarak gelen akışkan verileri de işleme ihtiyacının oluşmasından ötürü bu teknolojiler ortaya çıkmıştır. Yahoo!, gerçek zamanlı akışkan verileri işlemek için 2010 yılında S4'ü geliştirmiştir. Diğer şirketler ise günümüze daha yakın bir zaman diliminde kendi gerçek zamanlı veri işleme teknolojilerini geliştirerek kullanmaya başlamışlardır. Bunlara LinkedIn'in geliştirdiği Samza ve Google'ın geliştirdiği Millwheel örnek olarak gösterilebilir. Buradan da görülmekte olduğu gibi günümüzde gerçek zamanlı veri işleme teknolojileri gelişmeye devam etmekte, bu alanda duran veri işleme teknolojilerinde olgunlaşmış Hadoop gibi standart bir teknoloji henüz bulunmamaktadır.



Şekil 2.2. Büyük Veri işleme teknolojilerinin gelişimi

Üçüncü nesil veri işleme teknolojileri karşımıza 2012 yılından itibaren, hibrit veri işleme teknolojileri olarak Lamda Mimarinin tanımlanması ile çıkmaktadır. Çok yakın bir zamanda ortaya çıkmış olduğu için henüz teknolojik açıdan bile olgunlaşmamıştır. Gelecekteki yıllarda en iddialı teknolojilerden birisi olacağı düşünülmektedir [37]. Şekil 2.2'de Büyük Veri işleme teknoloji nesilleri gösterilmektedir.

2.5.1. Duran veri işleme teknolojileri

Duran veri işleme teknolojileri uygulama alanına göre farklılıklar göstermektedir. Duran veri işleme teknolojilerinden en olgun veri işleme yazılımı Hadoop'dur.

Hadoop Dağıtık Dosya Sistemi (Hadoop Distributed File System – HDFS), yapısal olmayan ve belirli bir formatı olmayan büyük miktardaki verilerin işlenmesi için kullanılmaktadır. Ancak ilişkisel veri tabanlarının yaygın olarak kullanımından ötürü, yapısal verilerin işlenmesinde de etkin olarak uygulanmaktadır. Veriler oldukları yerden komutlar ile HDFS'e kopyalanabilmektedir. Dağıtık dosya sistemine dosyalar yazılırken parçalara bölünerek birden fazla kopyası saklanacak şekilde gönderilmektedir. HDFS üzerinde farklı dosya formatlarında veriler saklanabilmektedir. Bu konuda bir kısıt bulunmadığı gibi analizlerde kullanılacak verilerin sıkıştırılması özelliği de mevcuttur. Java programlama dilinde geliştirilmiştir ve pahalı olmayan donanımlar üzerinde çalışacak şekilde tasarlanmıştır. Kümeleme şeklinde çalışarak yüksek veri transferi ve hatalara karşı dirençli ve ölçeklenebilir şekilde çalıştırılabilmektedir. HDFS sayesinde işlenecek dosyaların işlemeyi yapacak yazılım kodlarına değil, yazılım kodlarının dosyayı işlemek üzere bulunduğu düğüme gönderilmesi sağlanmaktadır. HDFS birden fazla düğüme efendi-köle (master-slave) mimarisi ile çalışmaktadır. HDFS üzerinde diğer Büyük Veri işleme ve saklama sistemleri de verilerini saklayabilmektedir. Büyük veriler ve kolonsal veri tabanları için veri tabanı dosyalarının saklanması için de kullanılmaktadır. Veriler HDFS'e bir defa yazılmakta ve birçok defa işlenmektedir. HDFS'e yazılan veriler sadece-okuma (read only) olarak kullanılmaktadır.

MapReduce Büyük Veri işlemede en yaygın olarak kullanılan programlama modelidir. Bu model, efendi-köle mimarileri üzerinde çalışan kümelemelerde dağıtık algoritmalar ve paralel veri işleme yöntemlerinden faydalanarak büyük miktarda verilerin etkin biçimde işlenebilmesini sağlamaktadır. Yapısında barındırdığı Map ve Reduce fonksiyonları, ufak iş parçacıkları olarak paralel şekilde dağıtık bir kümeleme sisteminde çalıştırılmaktadır. MapReduce modeli arka tarafta otomatik olarak haberleşme ve transfer işlemlerini kendi başına gerçekleştirmektedir. İşlenecek verinin olduğu yerde bırakılarak işleme ve analiz kodlarının veriye gönderilmesi oldukça büyük bir avantaj sağlamaktadır. İşlenecek verinin ağ üzerinden transfer edilmesi oldukça kaynak tüketen bir işlemdir. Map ve Reduce fonksiyonları anahtar-değer(key, value) ikilileri üzerinde işlem yapmaktadır.

Apache Sqoop, verinin ilişkisel veri tabanları gibi yapısal kaynaklardan toplu olarak HDFS'e aktarılmasını sağlamak için tasarlanmıştır. Ayrıca harici yapısal veri kaynaklarından HDFS ile ilişkili Hive ve HBase teknolojilerine veri aktarımı sağlayabilmektedir. Sqoop HDFS'deki verileri ilişkisel veri tabanlarına da toplu olarak yazabilme özelliğine sahiptir. Büyük Verilerin, etkin bir şekilde, Büyük Veri işleme sistemlerine ya da Büyük Veri işleme sistemlerinden veri tabanlarına aktarımını sağlamaktadır. Java programlama dili ile geliştirilmiştir. Yapılandırma dosyası üzerinden ya da API'ları yardımıyla veri kaynağına bağlanmakta ve MapReduce programlama modeli kullanarak verileri toplu olarak veri tabanlarından çekerek hedefe iletebilmektedir. Toplu ve yapısal veri kaynakları arasında veri transferini etkin bir şekilde gerçekleştirebilmektedir.

Apache Flume farklı amaçlar için kullanılabilen oldukça kullanışlı bir araçtır. Duran veri analizindeki en büyük katkısı veri edinme olarak gösterilebilir. Temelde büyük miktardaki akışkan veriyi güvenilir, ulaşılabilir ve dağıtık bir şekilde toplamak suretiyle, bir araya getirerek HDFS'e yazabilmektedir. Akışkan veriler üzerinde çalışabilen basit ve esnek bir mimariye sahiptir. Hatalara karşı düzeltme ve kurtarma mekanizmalarına sahiptir. Flume'un temel amacı veriyi toplayarak bir depolama birimine yazabilmektir. Mimarisinde üç eleman bulunmaktadır. Kaynaklar (sources) yeni gelen verileri toplamak için kullanılmaktadır. Yutak (sinks) kendisine gelen veriyi depolama birimlerine yazmak için kullanılmaktadır. Kanal (Channel) kaynaklardan gelen verinin yutaklara iletilmesi ve depolama birimine yazılınca kadar saklanması işlemlerini yürütmektedir. Kanallar dosya tabanlı ya da bellek tabanlı olarak kullanılabilir. Bellek tabanlı kanallar hataya duyarlıdır ancak hızlı çalışmaktadır. Dosya tabanlı kanallar ise, gelen veriyi dosya sisteminde saklamak suretiyle bir hata oluştuğunda Flume ajanını tekrar başlatarak veri iletiminin kaldığı yerden devam etmesini sağlamaktadır.

HBase; NoSQL tabanlı, ilişkisel olmayan, dağıtık yapıda çalışan, kolonsal bir veri tabanıdır. Google BigTable yaklaşımı kullanılarak geliştirilmiştir. HDFS üzerinde çalışmaktadır. HDFS'den farkı, temelinde kolonsal veri tabanı olması ve verilerin rastgele yazılıp okuna bilmesini sağlayabilmesidir. Verileri kolonsal bir yapıda sakladığı için doğal yollarla SQL sorgularına izin vermemektedir. Java programlama dili ile geliştirilmiştir. HBase, çalışma modeli olarak efendi-köle mimarisini kullanmaktadır. İlişkisel veri tabanlarına benzer şekilde her bir tabloda veriye erişmek için birincil anahtarlar

kullanılmaktadır. Tablolardaki birçok kolonun gruplanması ile kolon aileleri oluşturulmaktadır. İstendiğinde bir kolon, kolon ailelerine eklenebilmekte ya da kolon ailelerin çıkarılabilmektedir. HDFS'in üzerinde çalışmasından dolayı, verilerin hataya karşı korunması ve replikasyon gibi işlemleri HDFS'e bırakmaktadır. Ancak, HDFS ağ üzerinde gerçekleştirmekte olduğu haberleşme ve diğer işlemler nedeniyle veri tabanının sorgulara cevap vermesini geciktirmektedir. Veriler bölge sunucularında (region server) saklanmaktadır. Veri tabanında herhangi bir işlem uygulanmadan önce verinin hangi bölge sunucusunda olduğunun belirlenmesi gerekmektedir. Yapısında ayrıca hafıza tabanlı memcached anahtar-değer veri tabanını da barındırmaktadır.

Hive, Facebook tarafından geliştirilen bir veri işleme çatısıdır. MapReduce kullanılarak veri işlemek her zaman kolay bir işlem olmamaktadır. Ayır ayrı Map ve Reduce fonksiyonlarının geliştirilerek veriye uygulanması bazen veri işlemeyi zorlaştırmaktadır. Bu sorunu çözmek için farklı soyutlama (abstraction) teknolojileri geliştirilmiştir. Bunlardan bir tanesi de Hive'dir. Hive büyük veri işlemede SQL dilinin kullanılabilmesini sağlamaktadır. Hive temelde SQL dilinde yazılan sorguları Map ve Reduce fonksiyonlarına çevirerek verinin işlenmesini sağlamaktadır. Bir soyutlama katmanı olduğu için Hive ile yapılan veri işleme, doğal yollardan Map ve Reduce fonksiyonları kullanılarak yapılan veri işleme kadar etkin olmamaktadır. SQL kullanılarak veri işleme her zaman mümkün olmamaktadır. Özellikle makine öğrenimi tabanlı analizler ya da akışkan veri analizlerinde SQL kullanılarak veri işlenememektedir. Hive'da veri işlemek için verinin hangi formatta ve hangi bilgileri içerdiği, başka bir deyişle, yapısının bilinmesi gerekmektedir. Veri işleneceği zaman bir şema oluşturularak bu şema üzerinden ilgili alanlara ulaşılmakta ve veriler analiz edilmektedir. Hive'da SQL dilinin tüm yetenekleri desteklenememektedir. Ancak kullanıcı tanımlı fonksiyonlar (UDF) ile SQL desteğinin yeterli olmadığı durumlarda, kullanıcılar tarafından analiz fonksiyonları geliştirilmesine olanak sağlanmaktadır.

Pig, veri analizi için geliştirilmiş büyük veriler üzerinde çalışılmasına olanak sağlayan bir programlama platformudur. Programlar Pig-Latin adı verilen yüksek seviyeli bir dil üzerinde yazılarak bir derleyici vasıtası ile Hadoop üzerinde çalışacak MapReduce iş parçalarına dönüştürülmektedir. Hive bildirge (declaration) tabanlı bir dil kullanmakta olsa da pig izlek (prosedure) tabanlı bir programlama dili kullanmaktadır. Pig oldukça güçlü, esnek ve genişletilebilir bir yapıya sahiptir. Pig ile yazılan programların okunması ve

anlaşılması oldukça kolay olup bakım ve idamesinin yapılması da çok basittir. Ancak Hive ile aynı sorunları paylaşmaktadır. Bazı uygulamaların geliştirilmesi Pig ile hiç kolay değildir. Doğal (native) MapReduce uygulaması olarak çalışmadığı, Pig Latin ile geliştirilen algoritmaların sonradan Map ve Reduce'e dönüştürülmesi ile kullanıldığı için efektif değildir. Bazı birleştirme (Join) işlemlerinde Hive'dan daha düşük performans sergilemektedir. Kullanıcı tanımlı fonksiyonlar (UDF) ile Pig-Latin desteğinin yeterli olmadığı durumlarda kullanıcılar tarafından analiz fonksiyonları geliştirilmesine olanak sağlamaktadır.

Cascading; farklı türdeki veri işleme platformlarında çalışabilen, veri analitiği ve veri yönetimi uygulamaları geliştirilmesi için kullanılan bir Java uygulama geliştirme çatısıdır. Cascading Hadoop ve uygulama geliştirme arayüzü, Hadoop'a uyumlu tüm dağıtımlarla birlikte çalışabilmektedir. Karmaşık veri akışı işlemlerini MapReduce üzerinde soyutlama sağlayacak şekilde çalışan etkin bir katman olarak kullanılmaktadır. Buradaki yaklaşım Pig'deki yaklaşıma benzemektedir. Ancak Cascading ile oldukça karmaşık bir birini takip ederek çalışan iş akışı tabanlı analizleri gerçekleştirilebilmektedir. Pig ve Hive'e göre kullanımı daha zordur. Verinin bir biri ardına işleniyor olduğu durumlarda, her bir adım için MapReduce tabanlı fonksiyonların yazılması yerine kendi fonksiyonlarını kullanarak işlemleri kolaylaştırmaktadır.

Spark, Hadoop'tan sonra UC Berkeley tarafından geliştirilmiş ve günümüzün en revaçta Büyük Veri işleme platformudur. Yapısında birçok veri türünü işleyebilecek alt veri işleme bileşenlerini ve akışkan verileri işlemede kullanılabilecek platformları barındırmaktadır. Spark veri işlemede bellek kullanımı tabanlı bir yaklaşım kullanmaktadır. Yinelemeli algoritmaların geliştirilmesi için oldukça kullanışlıdır. Hadoop tabanlı teknolojilerde veri işleme tamamlandığında sonuçlar genellikle HDFS'e yazılmaktadır. Bir iş akışı uygulamasında bir sonraki adım için yazılan bu verilerin tekrar okunması ve işlenmesi gerekmektedir. Bu tip işlemlerde oldukça büyük gecikmeler oluşmaktadır. Bu tip işlemler Spark'ta daha farklı şekilde yapılmaktadır. Veriler işlendiği zaman bellekte saklanarak bir sonraki adımda kullanılmak üzere hazır halde tutulmaktadır. Spark, veri soyutlaması için RDD veri yapısını kullanmaktadır. Spark'ın sağladığı alt yapı; üst projeleri olan Spark SQL, Spark Streaming, MLlib ve GraphX tarafından kullanılmaktadır. Bu teknolojiler sırasıyla SQL tabanlı sorgulama yöntemleri ile verinin işlenmesini, akışkan verilerin işlenmesini, makine öğrenimi işlemlerinin gerçekleştirilmesini ve graf tabanlı analizlerin

yapılabilmesini sağlamaktadır. Spark, verinin saklanması için bellek, disk ya da her ikisinin de aynı anda desteklendiği bir elastik yaklaşımı benimsemektedir. Verinin hafızada saklanmasıyla veri üzerinde yinelemelere ve arttırımsal işlemlerin gerçekleştirilmesine olanak sağlamaktadır. Spark ayrıca sisteme çok fazla yük getirmeyen bir hata kurtarma sistemine de (lineage) sahiptir. Bir RDD kendi üzerinde bir bozulma yaşandığı zaman kendi kendini ve bağlantılı olduğu diğer RDD'leri yeniden oluşturabilecek düzeyde bilgiyi saklamaktadır. Spark iş parçacıkları birbirine bağlantılı bir akış şeklinde (DAG) saklanmakta ve işlenmektedir. Hesaplamalar gerçekleştirilirken tembel materyalizasyon (lazy materialization) yöntemi kullanılmaktadır.

Redis; bellekte çalışan, C programlama dilinde geliştirilmiş, karmaşık veri yapılarını destekleyebilen, anahtar-değer tabanlı bir veri tabanıdır. Yayınla/kayıt ol mesajlaşma haberleşme özelliğini desteklemektedir. Bellekte bulunan tüm verilerin diskte yedeklenmesini sağlayan görüntü alma (snapshotting) mekanizmasına sahiptir. Redis tek iş parçacığı (thread) üzerinde çalışmaktadır. Kendisine yapılan istekleri olay bilgilendirme stratejisi (event notification strategy) sayesinde asenkron olarak işlemektedir. Redis, sunucu tarafında betik kullanımını desteklemektedir. Kullanıcıların sunucu tarafında geliştirdikleri betikleri kullanabilmesini sağlayarak istemci tarafında tekrarlanan işlemleri sunucu içerisinde yapabilmelerini sağlamaktadır. Dağıtık yapıda yani kümeleme olarak çalışabilmektedir.

2.5.2. Akışkan veri işleme teknolojileri

Akışkan verilerin işlenmesi ve olduğu kaynaktan en hızlı şekilde işleneceği yere transfer edilmesi için çok sayıda, çeşitli ve olgun teknolojiler geliştirilmiştir. Büyük Veri işleme yaklaşımlarının duran veri kavramından akışkan veri işleme kavramına doğru kaymasıyla oldukça yaygın olarak kullanılmaya başlanmıştır. Veri daha henüz oluşmuşken işlenerek sonuçların gönderilmesi ya da veri transfer halindeyken işlenmesi gibi farklı yetenekte teknolojiler mevcuttur.

Apache Zookeeper, birbirinden farklı dağıtık sistemler arasında haberleşme ve iletişimi sağlayan hızlı koordinasyon servsidir. ZooKeeper sağladığı uygulama programlama arayüzü (API) ile farklı sistemlerin koordinasyonunu kendisi yapabilmektedir. Bu şekilde randevu, grup üyeliği, konfigürasyon yönetimi, basit kilitleme işlemleri, okuma yazma

kilitleri ve ikili bariyer gibi yeteneklerin kullanımını sağlamaktadır. Bir dosya sistemi mantığına benzer şekilde ad alanlarının (namespace) kullanımına olanak sağlayan basit bir mimariye sahiptir. Kullanıcılar bu ad alanlarının altındaki düğümlere okuma, yazma ve kitleme gibi işlemleri yapabilmektedirler. Bu yöntemler kullanılarak farklı dağıtık uygulamaların veri işleme, aktarım ve erişim işlemleri sırasında senkronize olmaları sağlanmaktadır.

Kafka, dağıtık ve ölçeklenebilir olarak çalışabilen mesajlaşma çatısıdır. Kendisine üretici (producer) ara yüzünden gelen mesajları başlıklar altında (topic) tüketicilere (consumer) gerçek zamanlı olarak iletmektedir. Başlık (topic) belirli bir tipte akışkan haldeki mesajları belirtmektedir. Üreticiler mesajlarını belirli bir başlık için yayınlarlar. Bu mesajlar simsarlar (broker) tarafından toplanarak depolanmaktadır. Simsarlara kayıt olan tüketiciler bir ya da daha fazla sayıda başlıktan bu mesajları kendilerine çekebilmektedir. Bir başlık, verinin paralel ve hızlı şekilde iletilebilmesi için bölütlere (partitions) ayrılmaktadır. Bölütler paralelleşmenin sağlanabildiği en küçük yapı birimidir ve simsarlar arasında mesaj yükünün dengelenmesini sağlamaktadır. Tüketici grupları, Kafka üzerindeki bölüt yapısına göre yükü dağıtabilmektedir. Kafka üzerinde depolanan bir mesaj mantıksal bir kayma (offset) ile belirlenmektedir. Bir tüketici, bölüt üzerinden gelen bir mesajı ardışık olarak kendisine alabilmekte ve mesajı aldığını bu kayma değerini belirterek bildirmektedir. Kafka üzerindeki simsar bir durum bilgisi ya da kendisine bağlanmış olan tüketicilerin durumu hakkında bir bilgi tutmamaktadır. Tüketici, kendisine gelen ve aldığı son mesajın kayma değerini saklamakla yükümlü olmaktadır. Bu özellik sayesinde tüketici, daha önce okunmuş ve kendisi tarafından alınan mesajları yeniden isteyebilme yeteneğine sahip olmaktadır. Uygulamalarda bir sıkıntı olduğunda, bu özellik sayesinde, istenilen mesajdan itibaren tüm mesajlar tekrar alınabilmektedir. Simsar ve tüketiciler birbirleri arasındaki haberleşme ve koordinasyonu ZooKeeper üzerinden gerçekleştirmektedir. Bu kapsamda, ZooKeeper yeni ya da artık kullanılmayan simsarların tespit edilmesine, başlıklara göre mevcut durumda başlık ve bölüt yapısına göre dengeleme yapılmasına ve tüketici gruplarının kendi kayma değerlerini saklamasına olanak sağlamaktadır. Kafka üreticileri kendi mesajlarını simsara senkron veya asenkron olarak gönderebilmektedir. Senkron mesaj gönderiminde bir defada sadece bir mesaj gönderimi gerçekleştirilmektedir. Asenkron mesaj gönderiminde Kafka, mesajları bellekte depolamakta ve depolama limitine gelindiğinde bir seferde hepsini iletmektedir. Senkron

gönderimin veri transferi daha yavaş olmakta, asenkron gönderimin ise hatalara karşı toleransı daha düşük olmaktadır.

Samza; YARN'ın (Yet Another Resource Negotiator) hataya karşı koruma, kaynak yönetimi ve iş izolasyonu yeteneklerinden faydalanan, dağıtık çalışan akışkan veri işleme çatısıdır. Görev dağıtımı ve mesajlaşma alt yapısı tamamen Kafka'nın üzerine kurulmuştur. Hatadan kurtarma, işlemci izolasyonu ve durum yönetimi gibi düşük seviyeli işlemler ve mesajların işlenmesi için basit bir API desteği bulunmaktadır. Bir Samza istemcisi, bir veya birden fazla akışkan veri işleme hattından gelecek mesajların alınması için işleme metodu geliştirmekte ve Samza, gelen mesajların yönetilmesini üstlenmektedir. Bir Samza iş parçacığı, veri işleme sonucunda elde ettiklerini bir Kafka akışına ya da harici bir veri depolama birimine yazabilmektedir. Samza, üzerinde periyodik ya da aperiodyik çizge (graf) bulunmasına göre çalışmaktadır. Samza iş parçacığı kendisine gelen mesajları işleyerek sonucu çıkış akış yoluna ileten mantıksal bir birimdir. Samza, veri işlemede paralelleştirmeyi sağlamak için veri yükü dağıtma işlemini Kafka bölütlerini kullanarak yapmaktadır. Bu şekilde, sadece bir bölütü kullanan görevler en küçük paralelleşmeyi sağlamaktadır. Mesajlar ardışık olarak geliş kayma değerlerine göre tüketilmektedir. Eğer bir görev, iki akışı birleştiriyorsa mesajlar round robin yaklaşımı kullanılarak tüketilmektedir. Kaynaklar kümeleme üzerinde çalışan YARN sayesinde dağıtılmaktadır. Eğer kaynak olarak YARN üzerinde Samza bölütü kadar konteyner yok ise bir konteyner üzerinde birden fazla iş parçacığı çalışacak şekilde ayarlanmaktadır. Bir düğümün çalışmaz hale gelmesi ya da YARN konteynerinde bir sorun oluşması durumunda Samza'nın hata giderme özelliği mevcuttur. Samza her bir iş parçacığı için Kafka üzerinde bir kontrol noktası başlığı (checkpoint topic) oluşturmaktadır. Samza bu başlığa en son işlediği verinin kayma değerini ayarlanabilir bir sürede yazmaktadır. Bir sorun yaşanması durumunda, Samza YARN konteynerini yeniden başlatarak son kaldığı kayma değerinden itibaren veriyi tekrar işlemektedir.

Storm; açık kaynak kodlu dağıtık çalışan bir gerçek zamanlı veri işleme sistemidir. Storm, veri işleme sürecini bir dizi çizge ile temsil etmektedir. Söz konusu çizge, Storm'un uygulama programlama arayüzü (API) kullanılarak tanımlanmakta, daha sonra Storm'un bulunduğu kümeleme üzerinde çalıştırılmaktadır. Storm kümelemesi, görev dağıtma işini yaptıktan sonra veri işleme işlemlerini yürütmekte, mesaj gönderimi ve alımı, hatadan kurtarma ve görev keşfi işlemlerini kendisi yürütmektedir. Storm geliştirilen topolojilerin

yürütülmesi işlemlerini Spout ve Bolt yardımı ile gerçekleştirmektedir. Akışkan veri kaynaklarına Spout ve veri işleme birimlerine Bolt adı verilmektedir. Akışkan veriler, bu sistemde alan adları kullanılarak, Tuple olarak modellenmektedir. Spout, bir dış kaynaktan veriyi alarak, topoloji içerisine Tuple olarak iletmektedir. Bir Spout güvenilir ya da güvenilir olmayan şekilde çalışabilmektedir. Güvenilir çalışan bir Spout; veri kurtarma, haberleşme sorunları ve verinin topolojiye tekrar gönderilmesi gibi problemleri kendisi çözebilmektedir. Güvenilir çalışmayan Spout ise bu yaklaşımı sergilemediği için veri kaybı yaşanabilmektedir. Bir Spout, topoloji içerisinde Tuple'larını birden fazla akış yoluna gönderebilmektedir. Bolt'lar, bir Storm topolojisinin işleme birimleridir. Bolt'lar; filtreleme, birleştirme, gruplama ve dış katmanlar ile haberleşme gibi birçok işlemleri üstlenmektedir. Storm topolojisinde bir akış oluşturulabilmesi için en az bir Spout ve bir Bolt kullanılması gerekmektedir. Bolt'lar veriyi işledikten sonra bir sonraki adımda bulunan Bolt'a veya Bolt'lara, bir ya da birden fazla akış yolu üzerinden gönderebilmektedirler. Storm Spot ve Bolt'ların paralelleştirme işlemlerini mesajların iletimi esnasında farklı şekilde gruplama yöntemleri kullanarak gerçekleştirmektedir. Bu gruplama yöntemleri karıştırarak gruplama, alan adlarına göre gruplama, küresel gruplama, direkt gruplama, yerel ya da karıştırarak gruplama ve tam gruplama olarak belirtilmektedir. Gruplama yöntemleri ile ihtiyaç duyulan davranışların sergiletilmesi ve paralelleşme sağlanması mümkün olmaktadır. Storm, topoloji içerisinde her bir Spout'tan gönderilen Tuple'ların Bolt'lar tarafından işlenmesini garanti etmektedir. Storm çalışırken bir Tuple ağacı oluşturarak, her bir Spout ve Bolt için Tuple'ların bu ağaç üzerinde akışının oluşturulması ile en az bir defa (at least once) iletim garantisi sağlanmaktadır. Bir mesaj işleme belirli bir zaman süresince tamamlanamadığında Tuple'ın işlemesi başarılı olmamakta ve daha sonra aynı Tuple yeniden işlenmek üzere gönderilmektedir. Bolt tarafından teslim bilgisi her bir Tuple için başka bir akış üzerinden gönderilerek o Tuple'ın başarılı şekilde işlendiği iletilmektedir. Trident, Storm'un üzerinde çalışan bir soyutlama katmanıdır. Akışkan veri işleme yöntemlerinin daha rahat ve kolay kullanılmasını sağlamak amacıyla geliştirilmiştir. Storm'da Bolt ve Spot'larda gerçekleştirilen işlemlerin daha yüksek bir düzeyde yapılmasını, geliştiricinin birçok karmaşık iş yükünden kurtulmasını sağlamaktadır.

S4; genel maksatlı, dağıtık, ölçeklenebilir, hataya karşı toleranslı ve genişletilebilir bir akışkan veri işleme platformudur. Gerçek zamanlı verilerin işlenmesi için Yahoo! tarafından geliştirilmiş bir teknolojidir. MapReduce tabanlı olarak geliştirilmesinden ötürü

ilk bakışta Storm'a benzerlik göstermektedir. Ancak hatadan kurtarma ve veri akışı mekanizmalarında farklılıklar bulunmaktadır. S4, mimari olarak klasik efendi düğüm yaklaşımını kullanmak yerine, merkezi olmayan simetrik mimari içerisinde, her bir düğümün benzer özellikte olduğu bir mimariyi kullanmaktadır. S4, kümeleme içerisinde ZooKeeper'ı koordinasyon katmanı olarak kullanmaktadır. S4'ün tasarımı MapReduce ve Aktör modelleri kullanılarak türetilmiştir. S4 üzerinde veriler, veri işleme elemanları tarafından işlenmektedir. S4, mesajları veri işleme elemanları arasında olaylar olarak(events) yayınlamaktadır. Her bir elemanın durumu (state) diğerleri tarafından erişilemez olup, hesaplama işlemleri veri elemanları arasında etkileşim ile gerçekleştirilmektedir. Bir S4 kümelemesi çok sayıda veri işleme elemanı içermektedir. Hatadan kurtarma işlemleri kısmi olarak desteklenmektedir.

Spark Streaming [40-41], duran veri işleme teknolojisi olarak kullanılan Spark'ın akışkan veri işleme için genişletilmiş olan teknolojisidir. Akışkan veri işleme işlemlerinin çok ufak toplu veriler üzerinde ardışık olarak gerçekleştirilmesine olanak tanımaktadır. Storm ve S4'den daha hızlı olduğu iddia edilmekte olup bir saniyenin altında bir gecikme sağlayabilmektedir. En önemli özelliği ise duran veri katmanında olduğu gibi akışkan veri katmanında da aynı programlama modelinin kullanılabilir olmasıdır. Gelen veri akışı bir dizi deterministik toplu hesaplama işlemlerine dönüştürülmektedir. Her bir parça kümeleme üzerinde güvenilir şekilde RDD [40] olarak saklanmakta ve durumsal ya da durumsal olmayacak şekilde işlenebilmektedir. Bir ara birim yardımı ile Kafka'dan mesajlar okunarak kümeleme üzerinde RDD'ler yardımıyla veriler işlenebilmektedir. Spark Streaming RDD'lerin değişmezlik (immutable) özelliğini ve lineage tabanlı hataya karşı toleranslı olma özelliğini bazı iyileştirme ve optimizasyonlar ile kullanmaktadır. Gelen veriler belirli bir zaman aralığı için bir dizi DStream adı verilen kesikli akışkan veriye dönüştürülmektedir. Yeni veriler geldikçe eski veriler bu veriler ile birleştirilebilmektedir. Akışkan veriler, kümeleme üzerinde iki farklı düğüme replike edilebilmektedir. Bir kümeleme düğümünün arızalanması ya da çalışmaması durumunda paralel olarak veri kurtarımı gerçekleştirilebilmektedir. Spark Streaming ile zaman penceresi tabanlı hesaplamalar gerçekleştirilebilmektedir. Akışkan veri ithali için Kafka, Flume, Akka, Actors, Raw TCP socketleri, HDFS ve MQTT'ye doğal desteği bulunmakta olup başka bir veri ithali oluştuğunda bu kaynağa özel alıcı çok kolay bir şekilde geliştirilebilmektedir.

2.5.3. Hibrit veri işleme teknolojileri

Hibrit hesaplama modelleri farklı katmanların varlığını gerektirmektedir. Bu modeli kullanan teknolojiler duran ve akışkan veri işleme modelini kullanan teknolojilere göre daha karmaşık yapıda olmaktadır. Her bir katman kendi uzman olduğu konuda veri işleyerek diğer katmanla koordinasyon içerisinde olmak zorundadır. Aynı anda iki farklı türde veri birbirinden bağımsız olarak işlenebileceği gibi, katmanların birbiri ile koordineli çalışması sonucunda, bir katmanın yapacağı hatanın diğer katman tarafından düzeltilmesi yaklaşımı benimsenmesi ile veri işlemek de mümkün olmaktadır. Bu alanda geliştirilmiş çok fazla teknoloji ve ürün bulunmamaktadır.

Lambdoop, bir Lamda Mimari oluşturmak amacıyla kullanılan birçok açık kaynaklı teknolojinin bir araya getirildiği bir soyutlama katmanıdır [42]. Büyük Veri uygulamalarını daha kolay bir hale getirmek için tasarlanmıştır. MapReduce benzeri bir model değildir. Pig ve Cascading gibi toplu veri işleme modeline benzerlik göstermektedir. Gerçek zamanlı veri işlemek için Trident'den daha kolay kullanılabilir. Lambdoop, tüm hibrit hesaplama modelini aynı programlama modeli ile birleştirmektedir. Lambdoop, bilgiyi veri nesneleri olarak akışkan ya da durağanlığından bağımsız olarak temsil etmektedir. Veri işleme işlemi, veri akışı ya da ardışık veri akışları üzerinde tanımlanan operasyonlar olarak tanımlanmaktadır. Akıllı ajanlar kullanılarak durağan, akışkan ya da hibrit veri işleme moduna göre verinin işlenmesi gerçekleştirilmektedir. Lambdoop henüz açık kaynaklı bir proje haline gelmemiştir ve geliştirilmesine devam edilmektedir.

SummingBird [43], Twitter tarafından geliştirilen Hadoop ve Storm üzerine inşa edilmiş bir kütüphanedir. MapReduce iş parçaları, duran veri için geliştirilerek Hadoop üzerinde çalıştırılmaktadır. Gerçek zamanlı akışkan veri için ise Storm kullanılmaktadır. Duran veri katmanında Hadoop, akışkan veri katmanında Storm kullanılmakta daha sonra sonuçların bir araya getirilmesi için birleştirme katmanı ile sonuçlar elde edilmektedir. Duran ve akışkan verilerin işlenmesi için belirli cebirsel yapıların entegre edilmesi yöntemi kullanılmaktadır. Summingbird'de analitik sorguların Hadoop iş parçaları ve Storm topolojilerinin programlama mantığı değiştirilmeden, Scala tabanlı, alana özel dil (DSL) kullanımıyla oluşturulması sağlanmaktadır. Ayrıca hibrit olarak çalıştırılarak duran ve akışkan sonuçların belirli zaman aralıkları içerisinde oluşturulmasını desteklemektedir.

Ancak Summingbird, sadece ve sadece kendisi için geliştirilen belirli cebirsel işlemlerle sınırlıdır. Veri işleme için yeterli esnekliği sağlayamamaktadır.

3. LAMDA MİMARİ VE TASARIMI

Bu bölümde Lamda Mimarinin temel özellikleri açıklanmaktadır. Lamda Mimarinin barındırdığı katmanlar ve bunların çalışması hakkında bilgi verilmektedir. Son olarak özgün olarak bir Lamda Mimarinin test edilmek üzere tasarımı yapılmıştır.

3.1. Lamda Mimari

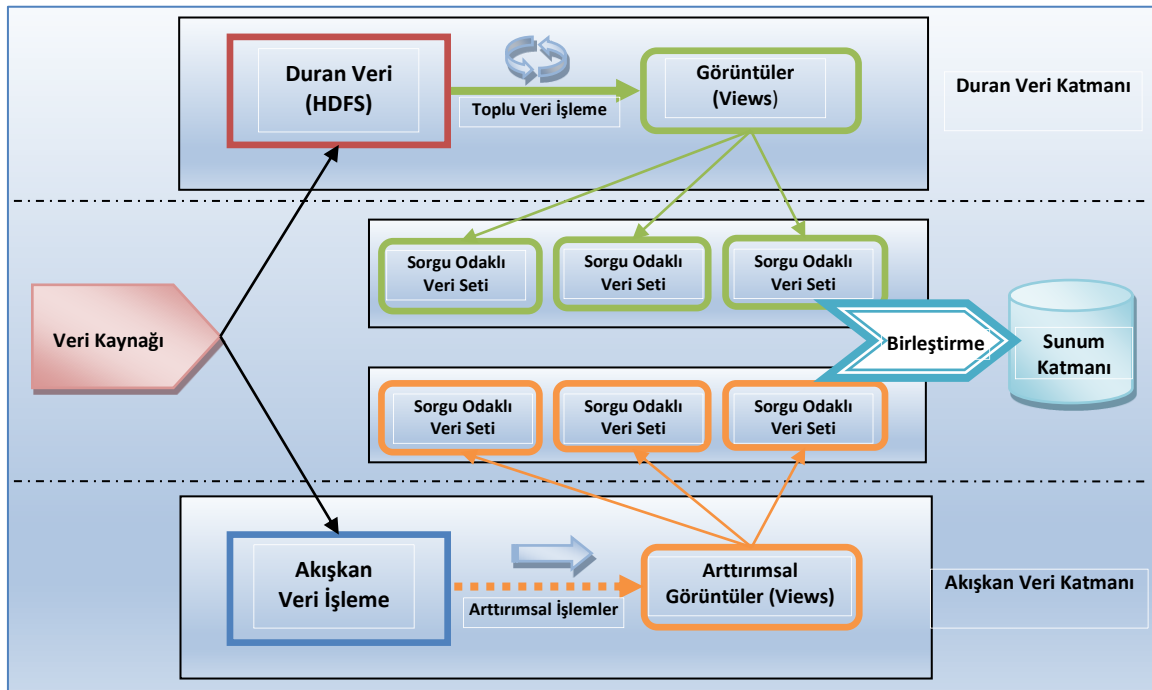
En temel anlamda düşünüldüğünde, bu veri tabanı sistemi geçmişten belirli bir zamana kadar bilgileri kullanarak sorgulara cevap vermektedir. Eğer bir veri sistemi geçmişteki verilere bakarak sorgulara cevap veriyor ise tüm verileri işlemek zorundadır. Bu tür bir sistem aşağıdaki gibi modellenenabilir [1]:

Sorgu = fonksiyon(Tüm Veri);

Lamda Mimari (LM), rastgele bir sistem üzerinde rastgele bir fonksiyon kullanımı ve bu fonksiyon ile düşük gecikmeli sorgular gerçekleştirilebilmesini sağlayabilmektedir. Bu yapı içerisinde, kesinlikle aynı teknolojilerin kullanılması yerine mevcut olan farklı teknolojilerin geniş bir şekilde bir araya getirilmesine olanak tanınmaktadır [1]. Ancak LM bu teknolojilerin seçilmesinde tutarlı bir yaklaşım sergilenmesini gerektirmektedir. Bu şekilde bir araya gelen ve birbirlerine bağlanan bileşenler tutarlı olarak çalışabilecektir. Bir Büyük Veri sistemi temel olarak aşağıdaki özelliklere sahip olmalıdır [1]:

- Tutarlılık ve hataya karşı dayanıklı olabilme
- Düşük gecikmeli (low latency) okuma ve yazma
- Ölçeklenebilir olma
- Genelleştirilebilme
- Genişletilebilme
- Anlık sorgulanabilme (ad-hoc)
- Minimum idame çabası gerektirme
- Hata ayıklayabilme

Yukarıda açıklanan bütün bu özelliklerin hepsini kendi bünyesinde barındıran bir sistemin gerçekleştirilemeyeceği düşünülse de Lamda Mimari, bu temel özelliklerin tamamını kendi yapısı gereği doğal olarak desteklemektedir. Rastgele fonksiyonların rastgele veri setleri üzerinde gerçek zamanlı olarak çalıştırılmasını destekleyen bir çözüm yoktur. Bunun yerine çeşitli araçlar ve teknikler kullanılarak eksiksiz bir Büyük Veri sistemi kurulabilir. Lamda Mimarideki temel fikir bir dizi katmanlı yapı ile bir Büyük Veri sistemi oluşturulmasıdır [1, 2]. Her bir katman, Büyük Veri sistemi için bazı gerekli özellikleri yapısında barındırarak bir alt katmanda bulunan yeteneklerin kullanılmasını sağlamaktadır. Şekil 3.1’de örnek bir Lamda Mimarinin mantıksal yapısı gösterilmiştir.



Şekil 3.1. Lamda mimari mantıksal yapısı

3.2. Lamda Mimari Katmanları

Şekil 3.1’den de görüldüğü gibi Lamda Mimari başlıca üç katmandan oluşmaktadır. Bunlar Duran Veri Katmanı, Akışkan Veri Katmanı ve Sunum Katmanı olarak adlandırılır. Söz konusu katmanlar tasarlanırken her birisi kendisine ait veri türünün işlenmesi için geliştirilmiştir ve veri türüne özgü teknolojiler kullanmaktadır. Bu şekilde diğer katmanlardaki verilerin türünden bağımsız olarak çalışılması ve birbirleri ile etkileşimde bulunulmaması sağlanmış olacaktır. AVK çalışma esnasında, DVK’a göre daha fazla yükte çalışmaktadır. AVK katmanına gelen verilerin geliş hızı ve miktarı oldukça büyük

olmaktadır. Gerçek zamanlı gelen bu verilerin işlenmesi ve gecikme olmadan sunum katmanına aktarılması gerekmektedir. Çok hızlı şekilde çalışmak zorunda olan bu katman diğerlerine nazaran daha fazla kaynak tüketmektedir. Bu katman sistem çalıştığı sürece aktif olarak görev yapmaktadır ve bu katmanın durması ya da çalışmaması durumunda gerçek zamanlı veri analizi sağlanamamaktadır. DVK ise aksine daha az bir yük altında çalışmaktadır. Ancak bu katmanın çalışma şekli kesiklidir. Belirli zamanlarda çalışarak analizini tamamlar ve analiz yaptığı kadar veriyi sunum katmanına iletir ayrıca sürekli veri katmanı üzerindeki yükü yine analiz yaptığı kadar veriyi silerek azaltmaktadır. Bu katmanın çalışmaması durumunda sistem ayakta kalabilir ancak analiz edilecek verinin veri kaynağı üzerinde herhangi bir nedenle gecikmesi durumunda ya da aynı verinin birden fazla sayıda iletilmesi durumunda oluşacak hatalar giderilemez. AVK'nın ne şekilde çalışacağı tasarım zamanında verilecek bir kararla netleştirilir. Tasarımda, "Belirli bir zaman penceresi içerisinde kalan veriler işlenir ve bu pencerenin dışında kalan zamanlarda (verinin gecikmesi durumu) veri göz ardı edilir işlemeye alınmaz." kararı alınabilir. Bu durumda analizler bir miktar eksik veri ile gerçekleştirileceğinden hata durumu oluşmaktadır. Bu hatayı gidermek tamamen DVK ile sağlanır. DVK belirli zamanlarda çalışarak geç gelen ya da tekrarlı gelen verileri düzelterek sunum katmanına iletir. Bu şekilde, oluşabilecek hatalar giderilmiş olur. Sistemin çalışma prensibine göre eninde sonunda güvenilir bir yapıda olacağı (eventually accurate) LM'nin geliştirilmesindeki esas kaidedir. LM'ler daima eninde sonunda güvenilir sistemler olarak kullanılmaktadır ve buna göre tasarlanmaktadır.

Sunum katmanı ise tam olarak AKV ve DVK'lardan gelen verilerin anlık olarak sorgulanabilmesine ve analiz sonuçlarının iletilmesine olanak sağlamaktadır. Bu katmanın tasarımında, büyük hacimlerde ve hızlarda gelen canlı verilerin dizinlenerek (indexing) sorgu odaklı görüntüler oluşturulması ve yine büyük miktarlarda kümelenmiş duran verilerin analizi sonucunda oluşan sorgu odaklı görüntülerin birleştirilerek kullanıcılar tarafından gerçekleştirilecek sorgulara kısa sürede cevap verebilmesi göz önüne alınmaktadır. Bu katmanın hem hızlı şekilde gerçek zamanlı veriyi ithal edebilmesi, hem de yüksek miktarda oluşmuş toplu veriyi hızla alarak kullanıma sunabilmesi gerekmektedir. Bu iki önemli özelliği aynı anda bünyesinde barındıran olgun bir teknoloji mevcut olmamasına rağmen, LM konseptini desteklemek için açık kaynak kodlu olgun olmayan ve gerçek üretim sistemlerinde kullanılması uygun olmayan çok basit çalışmalar mevcuttur. Bu nedenle çoğu firma LM'yi kullanılabilir hale girmek için kendi sunum

katmanlarını geliştirmektedirler. Literatür taramasında da belirtildiği gibi, bu durum NoSQL tabanlı farklı iki veri tabanının bir araya getirilerek aynı anda kullanılması gibi sonuçlara yol açmıştır. Bununla birlikte birçok akademik çalışmada sunum katmanına genellikle değinilmeyerek, bu konu üstün kötü geçirilmiştir. LM'yi ürün sistemlerinde kullanan firmalar ise çoğunlukla performansı düşük, işlem süresi uzun ve ölçeklenebilir olsa da hızlı sorgu sonuçları döndüremeyen NoSQL veri tabanları kullanılmaktadır.

Her bir katmanın belirli tasarım ihtiyaçları göz önüne alınarak geliştirilmesi gerekliliğinin yanında bir birlerinin hatalarını düzeltecek şekilde çalıştırılmalarının sağlanması gerekmektedir. Bu işlem basit şekilde sabit zamanlarda DVK'nın çalıştırılması şeklinde olabileceği gibi, AVK'nın üzerindeki yük belirli bir düzeye ulaştığında DVK'nın çalıştırılması şeklinde de olabilir. Bu çalışmada ikinci duruma göre tasarım yapılmıştır. Bu yaklaşımla tasarlanan LM'nin daha etkin, kararlı ve maliyet etkin çalıştığı belirlenmiştir.

3.2.1. Duran veri katmanı

Klasik sistemlerde ya da Büyük Veri sistemlerinde “Sorgu = fonksiyon(tüm data)” gibi bir yaklaşımla gerçekleştirilen sorgular mümkün olsa da çok uzun zaman almakta ve kaynakların gereksiz yere tüketilmesine sebep olmaktadır. Peta bytelar seviyesindeki bir verinin her seferinde bu şekilde sorgulanması durumu farklı bir yaklaşım geliştirilmesini gerekli kılmaktadır. Bu duruma alternatif olarak duran veri görüntüleri (batch views) kullanımı seçilebilir. Bu yaklaşımda önceden hesaplanan sorgu sonuçları toplu veri görüntüleri (DVG) olarak saklanır. Her defasında gerçekleştirilen sorgular tekrar tekrar çalıştırılmaz. Bunun yerine önceden hazırlanan DVG'ler sorgu sonucu olarak istek sahibine gönderilir [2, 29]. Böyle bir sistem aşağıdaki şekilde ifade edilebilir [1]:

DVG = fonksiyon(Tüm Veri);

Sorgu = fonksiyon(DVG);

Bu sistemde toplu işlenen veriler sonucunda elde edilen DVG'ler her defasında sorgu sonucu olarak hızlı bir şekilde ve kaynakları etkin kullanarak gönderilecektir. DVG'lerin üretilmesi yüksek gecikmeli (High-Latency) olmaktadır. Tüm veri üzerinde analiz

yapılacak ve işlem bittiğinde bu sırada birçok yeni veri gelmiş olacaktır. Dolayısı ile yeni veri sorgu sonucuna yansımayacaktır.

Toplu veri işleme katmanı temel olarak iki yeteneğe sahip olmalıdır [29]. Bunlardan birincisi, sürekli olarak büyüyen ve değişmezlik (immutable) özelliğine sahip olan temel veri setine sahip olması ikincisi ise herhangi bir hesaplama işleminin bu veri seti üzerinde çalıştırılabilir olmasıdır. Hadoop ve Spark [22] bu tür bir işlem için uygun Büyük Veri teknolojileridir. Toplu veri işleme katmanında geliştirilen kodlar sürekli olarak çalışmaya devam eder. Bu katmanda bulunan donanım sayısı artırılarak ölçekleme yapılabilir ve işlemlerin daha kısa sürede bitirilmesi sağlanabilir. Büyük Veri sistemleri tasarlanırken saklanan veri kullanılarak yapılacak analizlerde mümkün olduğunca fazla soruya cevap verebilmesi sağlanmalıdır.

Yeni analizler gerektiği durumlarda, ham verilerin tekrar işlenerek bu analizin gerçekleştirilebilmesi gerekmektedir. İşlenmiş veri, ham veriden türetildiği için içerisinde barındırdığı birçok bilgiyi kaybetmiştir. Bu nedenle Büyük Veri sistemlerinde olabildiğince ham verinin saklanması önemlidir. Verinin değişmez olması (Immutable) ile Büyük Veri sistemleri iki önemli avantaj elde eder. Bunlardan birincisi insan hatasına karşı verinin korunmuş olmasıdır. Verilerin yanlışlıkla veri tabanlarından silinmesi ya da güncellenmesi her zaman karşılaşılabilecek bir durumdur. Değişmez veri modelinde gereken tek şey yeni gelen verilerin ana veri kümesine eklenmesidir. Bu özellik sayesinde verilerin indexlenmesine gerek duyulmayacak ve ekstra işlem gerektirmeyecektir. Ancak bu model saklanacak veri boyutunun diğer veri saklama modellerine göre daha fazla yer kaplamasına neden olacaktır.

3.2.2. Akışkan veri katmanı

Sunum katmanı, toplu veri işleme katmanındaki hesaplama işlemi tamamlanır tamamlanmaz güncellenmektedir. Bu hesaplama ve veri işleme sırasında yeni gelen ve işlenmesi gereken veriler, toplu veri işleme katmanında işlenememektedir [29, 2]. Bu sorunun ortadan kaldırılması için, LM'ye eklenen akışkan veri işleme katmanında, toplu veri işleme sırasında gelen tüm verilerin gerçek zamanlı olarak işlenmesi sağlanmalıdır. Akışkan veri işleme katmanı sadece son gelen veriler üzerinde işlem yapmaktadır. Uygulamada ihtiyaç duyulacak ya da kabul edilebilecek gecikme zamanlarına göre gelen

veriyi artırımlı (incremental) olarak işlenmekte ve akışkan veri görüntüleri (AVG) oluşturmaktadır. Bu katmana ulaşan tüm veriler biriktirilerek sıfırdan işlenmemekte, bunun yerine AVG'ler artırımlı olarak güncellenmektedir. Bu katmanda yapılan işlem aşağıda gösterilmiştir [1]:

AVG = fonksiyon (AVG, Yeni Veri);

Yeni gelen veri ile halen üzerinde barındırdığı gerçek zaman görüntüleri artırımlı olarak güncellenmektedir. Dolayısı ile LM aşağıdaki şekilde özetlenebilir [1]:

DVG = fonksiyon (Tüm Veri);

AVG = fonksiyon (AVG, Yeni Veri);

Sorgu = fonksiyon (DVG, AVG);

Akışkan veri katmanı rasgele okuma ve yazma yapabilecek veri tabanları kullanmaktadır. Dolayısı ile kullanım ve çalışma açısından sunum katmanında kullanılacak veri tabanlarından çok daha karmaşık veri tabanlarının kullanılması gerekmektedir. Akışkan veri işleme katmanı diğerlerine nazaran daha karmaşık bir yapıya sahiptir. Toplu veri işleme katmanında işlenmesi tamamlanan veriler sunum katmanına aktarıldığında ve DVG'ler hazır olduğunda, akışkan veri görüntülerindeki bir miktar veriye artık ihtiyaç olmayacaktır. LM'nin sonuçları üzerinde geçici olarak saklayan bir katmana karmaşık işleri yaptırmasına karmaşıklık izolasyonu adı verilir. Akışkan veri işleme katmanında herhangi bir sorun olursa, toplu veri işleme katmanının bir sonraki çalışması ve sonuçları elde etmesi ile belirli bir süre sonunda bu sorun ortadan kalkacaktır. Artırımlı algoritma ve rastgele yazma erişimli veri tabanları kullanılması sebebiyle AVK, sunum katmanından ve DVG'dan çok daha karmaşıktır ancak LM'nin en büyük avantajı AVK'nın geçici (transient) yapıda olmasıdır [29, 2]. DVG ve sunum katmanları sürekli olarak AVK'nın ürettiği görüntülerin üzerine yazmaktadır ve AVK sadece belirli bir süreliğine veri saklamaktadır. DVG kendisine atanan veri işleme görevini tamamladığında AVK'nın ürettiği ve sunum katmanında bulunan AVG'nin bir miktarı silinmektedir.

3.2.3. Sunum katmanı

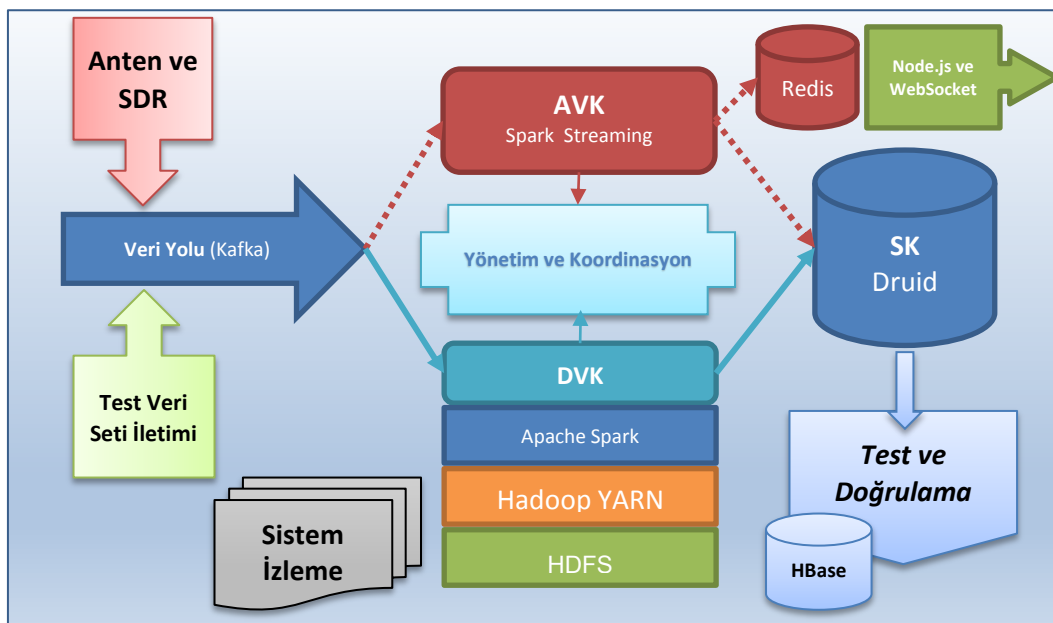
Toplu veri işleme katmanında hazırlanan DVG'lerin kullanılabilir hale gelmesi için bir sisteme yüklenmeleri gerekmektedir. Bu sistem ise sunum katmanının kullanılması ile oluşturulmaktadır. Bu katman dağıtık yapıda çalışabilecek, DVG'lerin yüklenebileceği ve rast gele okuma işlemlerinin gerçekleştirilebileceği bir veri tabanıdır. Yeni DVG'ler hazır olduğunda sunum katmanı eski DVG'leri devre dışı bırakmalı, yerine yeni DVG'leri kullanıma sunmalıdır. Sunum katmanı veri tabanı, toplu güncellenebilme ve rast gele okunabilme yeteneklerine sahip olmalıdır. Rast gele yazılabilme yeteneği veri tabanlarında en karmaşık işlemdir. Rast gele yazılma işlemini desteklememek bu karmaşıklığı ortadan kaldıracaktır. Bu şekilde basit çalışan veri tabanları kolayca konfigüre edilebilir, davranışı kestirilebilir ve kararlı çalışan bir yapıda olmaktadır [2, 29].

Sunum ve toplu veri işleme katmanlarının birlikte çalışabilmeleri ile rast gele veri setleri üzerinde rast gele sorgular çalıştırılabilmekte ancak DVG'ler oluşturulurken yeni gelen verilerin sunumu yapılamamaktadır. Bunların kullanılabilir hale gelmesi için bir miktar sürenin geçmesi ve toplu veri işleme katmanında çalışan kodun bir defa daha yürütülmesi gerekmektedir. Sunum ve toplu veri işleme katmanları bu şekilde kullanılsa bile düşük gecikmeli güncelleme yeteneği dışında Büyük Veri sistemlerinde olması gerekli özellikleri desteklemektedir.

3.3. Lamda Mimari Tasarımı

Bu bölümde lamda mimari bileşenleri, kullanılan teknolojiler ve birbirleri ile olan entegrasyonları ile her birinin çalışma prensipleri kısaca açıklanmıştır. Test verileri üretildikten sonra bu verilerin LM'ye iletilmesi gerekmektedir. Bu işlem için veri yolu katmanı kullanılmaktadır. Bu kısım genellikle LM içerisinde kabul edilmez, ancak mimarinin ayrılmaz bir parçasıdır. Şekil 3.3'de LM'nin tüm bileşenleri ve sistem gösterilmektedir. Sistemde test verilerinin ve gerçek zamanlı verilerin transferini sağlamak amacıyla veri yolu katmanı kullanılmaktadır. Veri yolu için Apache Kafka akış işleme teknolojisi kullanılmıştır. Veri yolu üzerinden aynı anda hem AVK hem de DVG veri ile beslenmektedir. AVK katmanında "Spark Streaming" kesikli akış işleme teknolojisi kullanılmıştır. Apache Kafka ve Spark Streaming, geliştirilen yazılım ile entegre edilmiştir. Spark Streaming, Kafka'ya bağlanıp verileri kendi üzerine alarak işlemektedir. Spark

Streaming ETL işlemini gerçekleştirdikten sonra analiz sonuçlarını elde ettiğinde bunları SK'da kullanılan teknoloji olan Druid'e göndermektedir. SK (Druid Realtime Dügümleri) ile AVK (Spark Streaming), "Tranquility" yazılım çatısı üzerinden entegre edilmiştir. Veri yolu (Kafka) kendisine gelen veriyi aynı zamanda DVK'ya iletmektedir. Bu iletim "Camus" teknolojisi üzerinden gerçekleştirilmiştir. DVK üzerinde YARN, Spark ve HDFS (Hadoop Distributed File System) servisleri çalışmaktadır. Gelen veriler belirli bir veri modeli kullanılmadan DVK'da HDFS'e yazılır. DVK'da Spark ile gerçekleştirilen analiz sonuçları SK'ya (Druid), SK katmanında bulunan, "OverLord" veya "Druid Hadoop Indexer" servisi ile gönderilmektedir. Gönderilecek veri büyüklüğü 1 GB'ı geçtiğinde "Hadoop Indexer", daha düşük hacimdeyse "OverLord" servisi kullanılmaktadır. Druid tarafından işlenen ve dizinlenen veriler yine HDFS'de saklanmaktadır. Druid'in, Spark'ın işlemesi sonucu oluşan duran verileri ithal etmesi işlemi ve zamanlaması yönetim ve koordinasyon katmanı tarafından AVK'nın kaynak tüketimine göre gerçekleştirilmektedir. Yönetim ve koordinasyon katmanında kullanılan tüm teknolojilerin birbirleri ile dağıtık olarak haberleşmesini sağlayan Apache ZooKeeper teknolojisi de kullanılmaktadır. Veri yolu, AVK, DVK ve SK yakın gerçek zamanlı olarak yönetim ve koordinasyon katmanı tarafından takip edilmektedir. Bu katmanla haberleşmede gecikmenin asgari düzeyde tutulması maksadıyla, AVK ve DVK üzerinde çalışan yazılımlarla doğrudan haberleşilmesi gerekmekte, Ganglia/Hadoop YARN üzerinde geliştirilen yazılımlar kullanılarak gerçekleştirilmektedir.



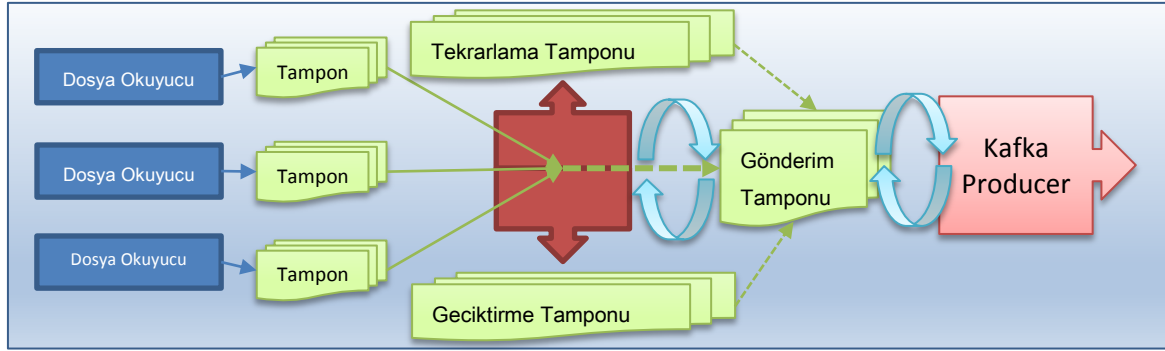
Şekil 3.2. Sistem Mimari ve bileşenleri

Sistem izleme yazılımı ise, Apache Flume ile AVK, DVK, SK, veri yolu ve test/doğrulama katmanlarını olay günlükleri vasıtasıyla takip etmektedir. Her bir katman gerçekleştirdiği işlemler süresince olay günlükleri oluşturarak, bunları Apache Flume ile sistem izleme katmanına göndermektedir. Bunlar Elastic Search ve Kibana yazılımı ile dizinlenerek (indexing) görselleştirilmektedir. AVK tarafından işlenen veriler gerçek zamanlı bir uygulamada kullanılacaksa Redis NoSQL veri tabanına Spark Streaming tarafından iletilmektedir. Redis veri tabanına gelen canlı veriler Node.js teknolojisi ile gerçekleştirilen bir uygulama ile WebSocket kullanılarak dış dünyaya servis edilmektedir.

Test ve Doğrulama katmanı, daha önceden MapReduce tabanlı algoritmalar tarafından işlenmiş sonuçların bulunduğu HBase veri tabanı ile HappyBase uygulama programlama arayüzü üzerinden haberleşerek test sonuçlarına erişmektedir. Druid uygulama programlama arayüzü üzerinden de ayrıca LM tarafından işlenmiş verilere ulaşarak sorgu sonuçlarını elde etmektedir. Geri Getirme ve Kesinlik Değerleri hesaplanarak JSON formatında bir dosyaya yazılmaktadır.

3.3.1. Veri yolu (Data Bus) katmanı

Veri yolu katmanında Kafka teknolojisi ve bu teknoloji kullanılarak geliştirilen bir yazılım kullanılmaktadır. Veri yolu katmanının görevi, gerçek çalışma durumunda, yazılım tabanlı telsizlerden gerçek zamanlı gelen verilerin alınarak AVK ve DVK'ya iletilmesi, bu işlem sırasında oluşan olayların (event) canlı olarak koordinasyon/yönetim ve sistem izleme katmanına gönderilmesidir. Ayrıca sistemin test edildiği durumlarda ise daha önceden toplanarak hazırlanmış veri setlerini kullanarak verilerin hızlı, ölçeklenebilir ve kontrollü bir şekilde AKV ve DVK'ya iletilmesini sağlamaktadır. Bu maksatla test veri setlerini okuyarak hafızada biriktiren, daha sonra bunu Kafka Producer kullanarak AVK ve DVK'nın ilgili entegratör teknolojilerine gönderen bir yazılım geliştirilmiştir. Söz konusu yazılım deney setlerinde bulunan verilerin dışında, testlerde kullanılacak tekrarlı veri üretimi ve geciktirme işlemlerini de gerçekleştirebilmektedir. Şekil 3.4'de geliştirilen yazılım mimarisi gösterilmektedir.



Şekil 3.3. Veri üretimi yazılımı

Veri üretimi yazılımı çalıştırıldığında yapılandırma dosyasından veri üretim parametrelerini okumaktadır. Söz konusu parametrelerle veri tekrarlama yapılması, verinin geciktirilmesi, veri gönderim hızı ayarlaması ya da normal olarak çalışması sağlanmaktadır. Yazılım çalışma şeklini belirledikten sonra öncelikle dosya okuyucu alt birimi vasıtasıyla her bir dosyadan verileri okuyarak veri okuma tamponuna (buffer) yazmaktadır. Veri okuma tamponu belirli bir düzeye geldiğinde, ayarlandığı çalışma şekline göre, buradaki verileri normal çalışma ise gönderim tamponuna yazmaya başlamaktadır. Gönderim tamponundan veriler okunarak, Kafka Producer ile Kafka Broker'a, yapılandırma dosyasında belirtilen veri gönderim hızı ile gönderilmektedir. Eğer veri geciktirmesi yapılacaksa okuma tamponundan gelen veri doğrudan gönderim tamponuna yazılmamaktadır. Geciktirilmesi istenen verilerin tarih ve saat bilgileri yine yapılandırma dosyasından alınarak, bu veriler süzülüp veri okuma tamponuna yazılmaktadır. Yine yapılandırma dosyasındaki bir parametre ile belirtilen zamanda veri gönderim tamponuna yazılmaktadır. Süzülen veriler dışındaki veriler normal çalışmada olduğu gibi doğrudan veri gönderim tamponuna yazılmaktadır. Tekrarlı çalışma durumu istendiğinde ise, veri okuma katmanından okunan ve yapılandırma dosyasında belirtilen tarih ve saat aralığındaki veriler, bu defa tekrarlama tamponuna yazılmakta ancak diğer veriler normal durumda olduğu gibi gönderim tamponuna iletilmektedir. Tekrarlama tamponundaki veriler konfigürasyon dosyasında belirlenen miktarda ve zamanda yine aynı dosyada belirtilmiş mesaj hızında tekrarlı olarak gönderim tamponuna yazılmakta ve Kafka Broker'a gönderilmektedir. Bu yapı ile birbirinden farklı zamanlarda hem verinin geciktirilmesi hem de tekrarlanması mümkündür. Veri üretici, üzerinde katı hal diskleri (SSD) bulunan tek bir bilgisayarda çalıştırılmaktadır.

Veri yolu katmanında kullanılan Kafka teknolojisi ile gelen yeni verilerin tamamı Kafka üzerinde saklanabilmektedir. Gelen verilerin tam olarak bir defa LM tarafından işlenmesi garanti edilmektedir. Kafka, bu yaklaşımla çalışacak biçimde ayarlanması durumunda kendisine gelen her bir veriyi bir günlük dosyasında biriktirmektedir. Kendisine gelen yeniden oynatma (replay) istekleri ile gerektiğinde aynı veriyi Kafka Broker vasıtasıyla göndererek sisteme tekrar iletilmesini sağlayabilmektedir. Kafka üzerinde saklanacak veri büyüklüğü, Kafka'nın veri saklama ayarları ile bir yapılandırma dosyası üzerinden ayarlanabilmektedir. Kafka Broker, bu şekilde kendisine gelen her veriyi ayarlandığı süre boyunca saklamakta ve gerektiğinde yeniden gönderebilmektedir. Ayarlanan süre tamamlandığında, veri Kafka servisi tarafından silinmekte, böylece aşırı veri birikmesi sonucu diskin dolması engellenmektedir. Bu özelliğin kullanım şeklinin tasarımın gerçekleştirildiği sırada belirlenmesi gerekmektedir.

Veri yolu katmanında, daha önceden kayıt edilmiş verilerin sisteme iletilerek LM'nin test edilmesini sağlayan veri üretici yazılımı bulunmaktadır. Aynı katman, yazılım tabanlı telsizlerle TCP protokolü kullanılarak socketler ile haberleşmekte ve gerçek zamanlı verileri Kafka Broker kullanarak LM'ye iletebilmektedir. Veri üretici yazılım, YKK ile doğrudan haberleşerek gerçek çalışma, test, geçmişe dönük verileri yeniden işleme şeklinde gönderilen görevleri yerine getirir. Test durumunda, YKK tarafından gönderilen komutlar ayrıca veri iletim hızının da canlı olarak ayarlanabilmesine olanak tanımaktadır.

3.3.2. Duran veri katmanı

DVK temel olarak canlı veri işleme durumunda oluşabilecek hataların düzeltilmesi amacıyla LM'de kullanılmaktadır. Duran veri katmanı, Kafka Broker üzerinden canlı olarak gelen verilerin, "Camus" entegrasyon yazılımı kullanılarak Hadoop dağıtık dosya sistemi HDFS'ye yazılması ile duran hale gelen veriler üzerinde çalışmaktadır. Camus, kendisine gelen veriyi HDFS üzerine yazarak biriktirmektedir. YKK katmanı tarafından DVK'nın çalıştırılmasıyla, belirli zamanlarda yeni gelen veriler işlenerek SK'da çalışmakta olan Druid'e iletilmektedir. DVK'nın çalışma sıklığı ve zamanı YKK tarafından yönetilmektedir. YKK saniyelik periyotlarda AVK ve DVK üzerindeki yükü ve veri akışını ölçmektedir. DVK, tasarım parametrelerindeki değerlere göre, AVK'ya gelen yeni verilerin kesintisiz olarak ithal edilmesi ile oluşan veri yükü, AVK'nın donanımsal olarak kaldıramayacağı seviyeye gelmeden, bu verilerin işlenmesi gerekmektedir. Ayrıca

DVK'nın bir defalık veri işleme süresinin, AVK'ya gelen yeni verilerin tamamen işlenmesi için geçen süreden küçük ya da bu süreye eşit olması gerekmekte ve bu işlemi geride kalmadan yapabilmesi zorunludur. Bu nedenle YKK tarafından yapılan saniyelik ölçümler ve kontrol algoritması ile belirli zamanlarda DVK çalıştırılmaktadır. ETL ve hesaplama işlemlerinin tamamlanmasından sonra yine YKK tarafından başlatılan Druid veri ithali işlemi ile yeni işlenen veriler SK'ya gönderilmektedir. Bu işlem eğer yeni gelen verinin boyutu 1 GB'ı geçmemişse Druid'in OverLord servisi ile yeni gelen verinin boyutu 1 GB'ı geçmişse Druid'in "Hadoop Indexer" servisi ile gerçekleştirilmektedir. Bu süreçte herhangi bir nedenle oluşan hatalar YKK tarafından tespit edilmekte, veri ithali yeniden yapılmakta ve DVK veri işleme işlemleri yeniden çalıştırılmaktadır. Arka arkaya üç defa hata alınırsa sistemin çalıştırılması tamamen durdurulmaktadır. Yeni gelen veriler veri yolu katmanında Kafka tarafından sürekli olarak biriktirilmeye devam edilmektedir. Kafka'da biriktirilen veriler Kafka üzerinde gerçekleştirilen bir dizi ayarlama ile istenilen süre boyunca saklanabilmektedir (Data Retention Policy). Sistem yeniden başlatıldığında, YKK tarafından Kafka Producer ile Apache ZooKeeper teknolojisi vasıtasıyla gerçekleştirilen haberleşme ve koordinasyon sonrasında aynı verilerin yeniden gönderilmesi sağlanabilmektedir. LM'nin en büyük kazanımlarından birisi olan bu özellik sayesinde, herhangi bir hata durumunda, sistemin eski zaman damgalı verileri tekrar işleyebilmesi mümkün olmaktadır.

DVK katmanında, YARN üzerinde çalışan Apache Spark teknolojisi etkin biçimde kullanılmaktadır. HDFS'e yazılan veriler belirli aralıklarla okunarak hafızaya alınmakta ve RDD olarak artırımlı (incremental) biçimde hafıza üzerinde biriktirilmektedir. Verilerin etkin biçimde işlenebilmesi için RDD'ler üzerinde yeniden dengeleme işlemi (repartitioning) YKK tarafından yapılan ölçümler sonucunda gerçekleştirilmektedir. Bu operasyon yapılmadığı takdirde sistemin işleyişi üzerinde ciddi performans kayıpları meydana gelmektedir. DVK'nın çalışması ve verinin SK'ya iletilmesi başarılı şekilde tamamlandıktan sonra, SK'da AVK'nın göndermiş olduğu ve DVK tarafından aynı zaman aralığında işlenen veriler boşaltılarak, AVK üzerindeki veri yükü azaltılmaktadır. DVK'nın çalışması tamamlandıktan sonra sisteme geciken ya da tekrarlı olarak gelen veriler nedeniyle oluşan hatalar tamamen düzeltilmiş olmaktadır.

Bu çalışmanın en özgün yanlarından birisi, AVK ve DVK'da farklı teknolojiler ile kodlama yapılması yerine aynı teknolojinin kullanılıyor olmasıdır. Aynı hesaplamaları ve

işlemleri yapacak yazılım kodunun AVK ve DVK için farklı teknolojilerde kodlanması yerine, bir defa kodlanarak kullanılması, iki farklı teknoloji için kod geliştirme, test etme ve bakım işlemlerinin ortadan kalmasını sağlamaktadır. Spark teknolojisi sayesinde geliştirilen yazılım kodlarının iki katmanda da aynı mantığı kullanılıyor olması nedeniyle bir defa kod geliştirilerek her iki katmanda kullanılmıştır.

YKK'dan gelen başlama komutları ile DVK, hafızada biriktirilmeye başlanmış ve başlama anına kadar olan tüm RDD'leri işlemeye başlar. Sonuçlar belirli zaman pencereleri içerisinde hesaplanarak HDFS'ye yazılmaktadır. Zaman penceresi, tasarım zamanında verilmesi gereken bir karardır. Büyük zaman pencereleri seçildiğinde, SK tarafından bu zaman pençeleri içerisinde verilerin ithal edilmesi uzun zaman alabilmekte bu nedenle SK'da bulunan sorgu odaklı görüntülerin oluşturulması ve güncellenmesi gecikebilmektedir. Küçük zaman pencereleri ise çok fazla sayıda ancak küçük boyutlu ithal dosyası oluşmasına ve aşırı işlemci tüketimine neden olabilmektedir. Bu nedenle uygulamanın gerektirdiği biçimde ve performansı en az etkileyecek şekilde zaman penceresi seçimi gerçekleştirilmelidir. DVK, çalışma sırasında gerçekleşen olayları (events) sistem izleme katmanına iletmektedir. Sistem, aynı zamanda YKK tarafından doğrudan YARN ve Ganglia yazılımı üzerinden takip edilmektedir. Tüm işlemler bittikten sonra kullanılmayacak RDD'lerin de ayrıca hafızadan temizlenmesiyle gereksiz yere hafızada yer kaplaması engellenmiş olmaktadır. Bu katmanda, ayrıca, üç farklı kaynaktan gelen verilerin aynı olduğunun belirlenmesi ve tek veri haline getirilmesi işlemi için bir füzyon algoritması da geliştirilmiştir. Gelen verilerin zaman damgaları ve koordinatlarının kullanılmasıyla aynı hava taşıtına ait olduğu belirlenmekte ve gelen bilgilerden zaman damgasına göre en son ulaşan veri üzerinde işlem yapılmakta diğerleri dikkate alınmamaktadır.

3.3.3. Akışkan veri katmanı

Akışkan veri katmanında yine bir Spark teknolojisi olan Spark Streaming modülü kullanılmıştır. Spark Streaming modülü, alıcı ara yüzü vasıtası ile Kafka Broker'da sunulmakta olan başlıklara (topic) kendini kayıt, ederek buradan kesikli olarak gelen verileri işlemek üzere alabilmektedir. Bu katman başlıca iki görev üstlenmektedir. İlk olarak veri yolundan aldığı bilgiler üzerinde zaman pencereleri oluşturarak, ETL ve analitik hesaplamalar yapmakta ve sonuçları arttırımsal olarak SK'da bulunan Druid veri

tabanına iletilmektedir. İkinci önemli görevi ise yine veri yolundan gelen verilerin vakit kaybetmeden gerçek zamanlı analitik uygulamasının veri tabanı olan Redis NoSQL'e yazılmasıdır.

AVK, kendisine üç farklı kaynaktan gelen tekrarlı verileri indirgeyerek tek bir veri haline dönüştürmektedir. Burada kullanılan füzyon algoritması, üç farklı kaynaktan gelen verilerin zaman damgasını ve koordinatlarını kullanarak aynı olan verileri belirlemekte ve zaman damgası en yeni olan koordinatları kullanmaktadır. Bu gereksinimden ve diğer ETL işlemlerinden ötürü AVK üzerinde kaynak tüketimi oldukça fazla olmaktadır.

AVK, Kafka Broker üzerinden gelen verileri Kafka uygulama programlama arayüzü (API) ile alarak bir araya getirmektedir. ETL işlemlerinden sonra söz konusu veriler gerçek zamanlı uygulamaya ve SK'ya gönderilmektedir. Sürekli biçimdeki veri, kesikli zamanlı verilere (DStreams) dönüştürülmektedir. Belirli zaman pencereleri içerisinde gelen veriler toplu olarak işlenmektedir. Bu nedenle zaman pencerelerinin uygulamaların ihtiyacına göre ve asgari gecikme oluşturacak şekilde tasarlanması gerekmektedir. AVK'da verilerin işlenmesi için 1s'lik zaman pencereleri seçilmiştir. Seçilen zaman penceresi süresince gelebilecek tekrarlı veriler de ayrıca tekil hale getirilerek iletilmektedir.

AVK'da işlenen verilerin SK'ya iletilmesi Druid Tranquility arabirimi ile gerçekleştirilmektedir. Druid veri tabanının Apache Kafka ve Apache Storm teknolojilerine doğrudan desteği olmasına karşın, henüz Spark için bir desteği bulunmamaktadır. Bu nedenle Spark ile Druid arasındaki veri iletimini sağlayabilmek için Tranquility yazılım çatısı kullanılarak bir arabirim geliştirilmiştir. Bu arabirim vasıtasıyla gelen tüm veri Spark Streaming ile işlendikten sonra kayıpsız olarak SK'ya iletilebilmektedir. Tranquility yazılım çatısı vasıtasıyla Druid veri tabanına iletilen tüm veriler gerekli performans indikatörleri hesaplanarak gerçek zamanlı olarak yazılmaktadır. Spark Streaming ile SK'ya gönderilen veriler anahtar, değer ikilisine (Key,Value) dönüştürülerek Redis veri tabanına iletilmektedir. Redis veri tabanı yayımla-üyelik modeline göre çalışmaktadır (publish-subscribe). Bu nedenle bir araya getirilen DStream RDD'lerdeki verilerin doğrudan Redis veri tabanına yazılması için Redis uygulama programlama arayüzü (API) kullanımından başka özel entegrasyona ihtiyaç duyulmamıştır.

AVK kendisine ait işlemleri gerçekleştirirken meydana gelen olayları (events) Log4J2 yazılım çatısı ve Apache Flume teknolojisi ile gerçek zamanlı olarak sistem izleme katmanına iletmektedir. Ayrıca Spark Streaming için çok önemli metrikler olan işleme zamanı ve zamanlama gecikmesi (processing time ve scheduling delay) sistem izleme katmanına ve YKK katmanına aynı anda gönderilmektedir. Bu iki metrik kullanılarak DVK'nın çalıştırılmasına karar verilmektedir.

3.3.4. Sunum katmanı

Sunum katmanı Lamda mimarinin en önemli, teknolojik açıdan da gerekli yetenekleri en az karşılanan bölümüdür. Kullanımı, entegrasyonu ve idamesi oldukça zor olan bu katman hakkında literatürde yeteri kadar çalışma olmadığı hatta hep göz ardı edildiği tespit edilmiştir. Yine literatür taramalarında bu katmanın iki farklı veri tabanı teknolojisi ile gerçeklenmeye çalışıldığı görülmüştür [30, 33-34]. Bu durum iki farklı veri tabanından sorgulama yapıp, sonuç olarak oluşan verilerin bir araya getirilmesini sağlayacak bir yazılımsal müdahaleyi gerekli kılmaktadır. Bu katman bir Büyük Veri sisteminin sorgu katmanı olduğu için, iki farklı sorgu sonucunun birleştirilmesi aşamasında oldukça sıkıntılar yaşanmaktadır. İşlenen verilerin gerçek zamanlı ve geçmişe dayalı olarak aynı anda sorgulanması gerekliliği bu katmanın varlığının en büyük sebebidir. Ancak teknolojik açıdan Lamda Mimarinin teknik ihtiyaçlarını karşılayacak yetenekler, olgunlaşmış ve kararlı biçimde çalışan bir teknoloji ile desteklenememektedir. Bu katmanın, gerçek zamanlı olarak gelen verileri, gecikmeden ithal edebilecek ve ithal işlemi gerçekleştiği anda sorguya hazır hale getirerek gerçek zamanlı sorgu odaklı görüntüler oluşturabilecek, bu işlem gerçekleşirken sürekli olarak atomik/atomik olmayan okuma ve yazma isteklerini destekleyebilecek bir yapıda olması gerekmektedir. Ayrıca, duran verilerin hızlı bir şekilde sadece sorgu odaklı görüntülerini oluşturacak şekilde toplu ithal işleminin de DVK katmanında desteklenmesi gerekmektedir.

Farklı veri türlerinden sorgu odaklı görüntüler (query focused views) oluşturulması gerekmektedir. Doğru teknolojinin doğru veri türünde kullanılması ve sorgular için entegre edilmesi her ne kadar doğru bir yaklaşım olsa da, büyük verilerle uğraşılacak bir sistemde etkin bir yaklaşım olmayacaktır. Büyük verilerin farklı katmanlardan sorgulanması ve bu farklı katmanlardan gelen verilerin birleştirilerek sorguya cevap verilmesi oldukça kaynak tüketen ve gecikme yaratabilen bir durumdur.

Yukarıda belirtilen sebeplerden ötürü, sunum katmanının tasarımı, kullanımı ve farklı veri katmanlarına entegrasyonu üzerine araştırmalar gerçekleştirilmiş ve en doğru teknolojinin Druid [44-45] olduğu belirlenmiştir. Ancak Druid bazı gereksinimleri karşılıyor olsa bile tam olarak istenilen düzeyde ihtiyaçlara cevap vermemektedir. Açık kaynaklı olan bu teknoloji üzerinde kaynak kodları kullanılarak geliştirmeler/değişiklikler yapılarak Lamda Mimaride sunum katmanında kullanılmıştır. Bu şekilde farklı veri türleri için sorgu odaklı görüntülerin oluşturulması ve en iyi performansın yakalanması sağlanmıştır.

Bu tür özelleştirilmiş bir sistemin AVK ve DVK'dan sürekli olarak veri ithal edebilmesi, bu süreçte hiçbir veri kaybına neden olmaması ve eksiksiz sorgulanabilir şekilde çalışması gerekmektedir. Bu nedenle SK'nın tüm bu isterleri gerçekleştirmesi için farklı bileşenlerden oluşması ve bu bileşenlerin birbirleri ile koordineli olarak çalışması gerekmektedir. SK temel olarak dört alt bileşenden oluşmaktadır. Bunlar duran veri düğümleri, akışkan veri düğümleri, koordinatör düğümler, akışkan/duran veri ithal servisleri ve aracı düğümlerdir. Tüm düğümler bir kümeleme (cluster) içerisinde çalışmaktadır.

Duran verilerin ithal edilmesi ve dizinlenmesi, bölütlerin (segment) oluşması ve gelen sorgulara cevap verilmesi gibi işlemler duran veri düğümleri (historical nodes) yardımıyla gerçekleştirilmektedir. Bu sistemde bölütler (segment) sorgu odaklı görüntüleri temsil etmektedir. Verilerin hangi katmandan gelirse gelsin en hızlı şekilde ithal edilmesi, analitik hesaplamalarının gerçekleştirilmesi ve gerekli zaman çözünürlüğünde bölütlerin oluşturularak dizinlenmesi, bu süre zarfında da sorgulara cevap verilmesi duran veri düğümü sayesinde gerçekleştirilir. Bölütler, sisteme veri ithali ve dizinleme yapılması için kullanılan "OverLord" servislerinde zaman penceresi, metrik ve boyutlarının ayrıca veri formatlarının tanımlanması ile oluşturulur. Duran verilerin sisteme ithal işlemi hem OverLord hem de HadoopDruidIndexer MapReduce iş parçası ile gerçekleştirilebilmektedir. Veri ithali sonucunda bu düğümlerde oluşan bölütler başka bir değişle sorgu odaklı görüntüler değiştirilemez (immutable) yapıdadır. Daha önce de belirtildiği gibi bu Lamda Mimari DVK'sının temel özelliğidir. Bu düğümler "hiçbir şey paylaşma" (shared nothing) prensibine göre çalışmakta olup, bölütleri sorgulamalar için kullanıma sunma, kullanımdan çıkarma ve sorgulara yanıt verme işlemlerini kendi başına sağlamaktadır. Belirli bir zaman penceresi için ve belirli bir zaman çözünürlüğünde oluşturulan bölütlerin tamamlanmasıyla, bu bölütler derin saklama (deep storage) alanlarında kullanıma sunulmaktadır. Druid bu alanlarda HDFS ve Apache Cassandra

teknolojilerini kullanabilmektedir. Kolonsal formatta hazırlanan bölütler, derin saklama alanlarından DVK düğümlerine indirilmekte ve aracı (broker) düğümlerden gelen sorgu cevapları yine aracı düğümlerine iletilerek cevap verilmektedir. Duran veri düğümleri sorumlu olduğu zaman penceresindeki bölütler ile ilgili bilgileri Apache ZooKeeper teknoloji kullanarak diğer düğümlere iletmektedir. Benzer şekilde, yine Apache ZooKeeper üzerinden gerçekleştirdikleri koordinasyonlar ile üzerinde bulunan bölütlerin kullanıma alınması ve kullanımdan kaldırılması işlemlerini de gerçekleştirmektedir.

Aracı düğümler (nodes) istemciler tarafından gerçekleştirilen sorguları akışkan veri ve duran veri düğümlerine iletmektedirler. Aracı düğümler, sorgu sonuçlarını aldıklarında bu sonuçları birleştirmekte ve sorguyu gerçekleştiren istemcilere iletmektedirler. Aracı düğümler, kümelemede (cluster) hangi akışkan veri ve duran veri düğümlerinin olduğunu Apache ZooKeeper üzerinden takip etmektedirler. Sorgulama işleminde bölütler sorumlu oldukları zaman pencerelerine göre farklı düğümlerde bulunmaktadır. Bu nedenle, gelen sorguların doğru düğümlere yönlendirilmesi için, aracı düğüm, öncelikle ZooKeeper kullanarak akışkan veri ve duran veri düğümleri ve bu düğümlerim sahip olduğu bölütleri belirlemektedir. ZooKeeper’da bulunan her bir veri kaynağı için, aracı düğüm, bir düğüm zaman cetveli (timeline) ve bu bölütlere ait düğümleri belirlemektedir. Belirli bir veri kaynağı ve zaman penceresi için sorgu aracı düğüme ulaştığında, aracı düğüm, verilerin bulunduğu düğümleri belirlemekte ve sorguları bu düğümlere yönlendirmektedir.

Akışkan veri düğümleri, veriyi gerçek zamanlı olarak ithal etmektedir. Gelen veri akışını dinlemekten ve veri tabanına yazıldığı anda derhal kullanıma sunmaktan sorumludur. Bu düğümler ayrıca aracı (broker) düğümlerden gelen sorgu sonuçlarına da cevap vermektedir. Belirli bir süre boyunca biriken veri, akışkan veri düğümünden derin saklama alanlarına gönderilmektedir. Akışkan veri düğümleri, oluşmalarından sonra derin saklama alanlarına gönderilen bu bölütlerin, duran veri düğümlerine (historical nodes) yüklenmesi ve kendi üzerindeki duran veri düğümlerinde sunulmaya başlanmış bölütlerin silinmesi işlemlerini de gerçekleştirmektedir. Bu işlem için yine ZooKeeper kullanılmaktadır. Akışkan veri düğümlerinde veri ithali iki şekilde gerçekleşmektedir. Bunlardan birincisi kendi başına çalışan (stand alone) gerçek zamanlı düğümlerin kullanılması, ikincisi ise dizinleme servisi olarak Tranquility istemci kütüphanesinin kullanılmasıdır. Kendi başına çalışan düğümler ile yüksek performanslı sonuçlar alınamadığı ve bazı kısıtlar ile karşılaşıldığı için Tranquility istemci kütüphanesi kullanılmıştır. Gerçek zamanlı verinin

ithal edilmesi bir yapılandırma dosyası yardımıyla gerçekleştirilmektedir (druid.realtime.specFile). Söz konusu dosya JSON dizilerden oluşan bir metin dosyasıdır.

Koordinatör düğümler, duran veri düğümlerindeki verilerin sunuma hazır hale getirilmesi, replikasyonu ve en optimal şekilde çalışması için duran veri düğümlerini izlemektedir. Bu düğümler, ZooKeeper üzerinden duran veri düğümlerinin bulunması, gerekli bölütlerin kümelemeye yüklenmesi ve gerek duyulan bölütlerin duran veri katmanlarına yüklenmesi/silinmesi işlemlerinin başlatılması işlemlerini metaveri bilgisinin takip edilmesi ile gerçekleştirebilmektedir. Temel olarak bölüt düzenlemesi ve dağıtımından sorumlu olmakta, duran veri düğümleri ile haberleşerek gerekli bölütlerin yüklenmesi ya da silinmesi işlemlerini gerçekleştirmektedir. Ayrıca bölüt replikasyonu ve bölüt dengelemesi işlemlerini de gerçekleştirmektedir. Koordinatör düğümü ayarlanabilir bir parametre kullanılarak periyodik olarak belirli zaman aralıklarında çalışmakta ve bir işlem gerçekleştirmeden önce kümelemenin (cluster) o anki durumunu değerlendirmektedir. Daha sonra kümeleme durumuna göre yapılması gereken işlemleri gerçekleştirmektedir. Koordinatör ZooKeeper bağlantısının dışında, kümelemede bulunan bölüt ve kurallar için bir veri tabanı ile bağlantı kurmaktadır. Bu veri tabanında kullanımda olan bölütler bir tabloda saklanmakta ve kümelemeye yüklenmesi gerekli bölütler de listelenmektedir. Kurallar ise bölütler üzerinde hangi işlemlerin yapılması gerektiğini belirtmektedir. Henüz kümelemeye dâhil edilmemiş bölütler için sunucuların kapasitelerine göre yükleme önceliği belirlenmektedir. Koordinatör duran veri düğümleri ile doğrudan bağlantı kurmamaktadır. Bunun yerine koordinatör düğüm yüklenmesi gereken bölütler için geçici bir bilgiyi duran veri düğümünün kuyruk (queue) yapısına iletmektedir. Bu istek duran veri düğümü tarafından alındığında, bölüt derhal yüklenmekte ve kullanıma sunulmaktadır.

Dizinleme (Indexing) servisi, dizinleme ile ilgili görevlerin gerçekleştirildiği, dağıtık çalışan, yüksek erişimli bir servistir. Dizinleme servisi görevler oluşturarak bölüt dosyaları oluşturmakta, bazen de silmektedir. Çalışma yapısı olarak efendi/köle (master/slave) benzeri bir mimari kullanmaktadır. Bu servis temel olarak üç farklı bileşenden oluşmaktadır. Bunlar tek bir görevin çalıştırıldığı “Peon”, Peonların yönetildiği orta katman yönetim birimi ve görev dağıtımının gerçekleştirildiği “Overlord” bileşenleridir. OverLord ve orta katman yönetim birimleri aynı düğüm üzerinde çalışabildiği gibi birden fazla düğümde de çalışabilmektedir. Ancak orta katman yönetim birimleri ile Peon’lar her zaman aynı düğüm üzerinde çalışmaktadır.

OverLord düğümü gelen görevlerin kabul edilmesi, görev dağılımının koordine edilmesi, görevler arasında kilit sisteminin kullanılması ve görev sonuçlarının bildirilmesi işlemlerinden sorumludur. Yerel ya da uzak çalıştırılma yöntemlerini desteklemektedir. Yerel çalıştırılmada OverLord, işlerin tamamlanması için Peon yaratılmasından da sorumludur. Uzak çalıştırılmada ise, OverLord ve orta katman yönetim birimi farklı iş parçacıkları içerisinde çalışmaktadır. Orta katman yönetim düğümü, kendisine gönderilen görevlerin çalıştırılmasını sağlayan bir işçi (worker) düğümdür. Gelen görevleri farklı bir java sanal makinesi JVM'de çalışmakta olan Peon'lara göndermektedir. Her bir Peon, bir defada sadece bir görev tamamlayabilmektedir ancak orta katman yönetim birimi birden fazla Peon yürütebilmektedir.

Verilerin ithal edilmesi ve işlenmesi için veri şemaları kullanılmaktadır. Gelen veriler "FireHose" arayüzü üzerinden belirtilen veri ithal etme şemaları ile ithal edilmekte ve dizinlenmektedir. Veri şemaları, gelen veriler üzerinde gerçekleştirilecek bir araya getirme işlemleri, kullanılacak alanlar ve gerçekleştirilecek hesaplamalar ile ilgili bilgileri tanımlamakta olan birer JSON metin dosyasıdır. Şemalarda temel olarak kullanılan bilgiler genelde üç türlü olmaktadır. Veri boyutları (dimensions), verilerin bir araya getirilmesi için metin tabanlı etiketlerdir. Bu etiketler vasıtasıyla veri üzerinde tanımlanan şemadaki sayısal değerler, verilen istenen fonksiyonlarına göre bir araya getirilerek (aggregation) hesaplanmaktadır. Zaman damgaları ise veri şemalarının en önemli birimi olup kesinlikle boş bırakılamaz. Gelen veriler veri şemasında belirtilen zaman damgası alanlarına göre ithal edilmekte ve işlenerek dizinlenmektedir. Gelen her verinin yapısında zaman damgası farklı alanlarda bulunabileceği için şema içerisinde zaman damgası alanı net bir şekilde belirtilmektedir. Veri şemalarında, ithal edilecek veri yapılarında sayısal değerlerin tutulduğu ve bu alanlar üzerinde sayma, minimum/maksimum bulma, toplama, bölme vb. gibi işlemlerin gerçekleştirildiği veri alanları metrik adı verilen bölümlerde belirtilmektedir. İthal edilen gerçek zamanlı ya da duran veriler üzerinde yapılacak hesaplama işlemleri için sayısal değerleri temsil etmektedirler.

Veri şemaları içerisinde ayrıca gelen verinin kaynağı ve formatını belirten (DataSource ve Parser) bilgi alanları da bulunmaktadır. Gelen verilerin JSON, virgülle ayrılmış değerler (CSV), XML gibi formatlarda geldiğini belirtmek, ayrıca gerçek zamanlı ya da duran veri kaynağından verinin ithal edileceğini göstermek amacıyla kullanılmaktadır.

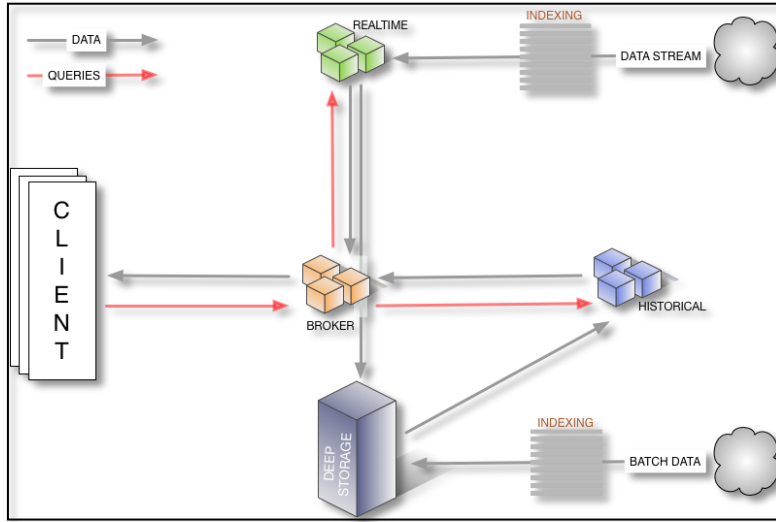
Gelen veriler üzerinde işlem yapılacağında, bölütlerin oluşturulması için bir araya getirme işlemi (aggregation) zaman damgaları kullanılarak çözünürlük (granularity) alanları yardımıyla gerçekleştirilir. Şemadaki çözünürlük değerlerine göre (1 dk., 10 dk., 30 dk., 1 gün ..vb.) veriler ithal edildiği anda bir araya getirilmekte ve şemada belirtilen matematiksel fonksiyonlar metrik alanlarına uygulanarak kayıt edilmektedir. Bu nedenle veri tabanında gerçek veri tutulmadığı için tasarım zamanında veri kullanım çözünürlüğüne karar verilmesi gerekmektedir. Düşük zaman dilimli çözünürlükte bir araya getirilmiş veriler daha yüksek zaman dilimli çözünürlüklerde kullanılabilmekte ancak daha yüksek zaman dilimli çözünürlüklü veriler, daha düşük zaman dilimli çözünürlüklü sorgularda kullanılamamaktadır. Bu, analitik veri tabanlarının en önemli özelliğidir. Veri ithal edildiği anda bir araya getirilerek sadece daha sonra talep edilecek ya da kullanılacak sorgulara cevap verebilecek şekilde kayıt edilmektedir. Ham veri onar dakikalık çözünürlükte ithal edildiğinde, birer dakikalık sorgulara cevap veremeyecektir. Ancak yarım saatlik, haftalık, yıllık zaman dilimlerinde talep edilen sorgulara cevap verebilecektir.

Derin saklama alanları (Deep Storage) bölütlerin depolandığı alanlardır. Bu özelliği Druid kendi başına sağlamamaktadır. Derin saklama mimarisi verinin saklanma süresini tanımladığı gibi, bölütler üzerinde sorgular çalıştırılmasına da olanak tanımaktadır. Bölütler bu alanlardan kaybolurlarsa, bu bölütlerdeki veriler de kaybedilmiş olacaktır. Şu an için Druid yerel diskleri, HDFS'i, Amazon S3'ü, Apache Cassandra'yı ve Microsoft Azure teknolojilerini derin saklama alanları olarak kullanabilmektedir.

Metaveri kullanımı ve saklaması için Druid şu an Derby, MySQL ve PostgreSQL veri tabanlarını kullanabilmektedir. Kullanılacak veri tabanı teknolojisi yapılandırma dosyasında belirtilerek aktif hale getirilmektedir. Bu veri tabanında bölütler, tablolar, kurallar, izinleme görevleri ile ilgili tablolar, erişim zamanları ve değişiklik yapan kullanıcı bilgilerinin saklandığı tablolar mevcuttur.

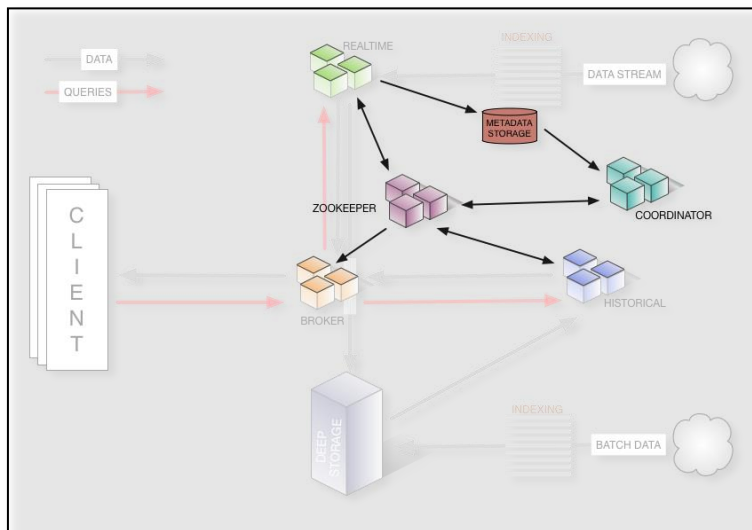
Sunum katmanında her bir göreve yönelik gerçekleştirilmesi gereken işlemlerin farklı donanımlar üzerinde çalışacak şekilde farklı düğümlerde icra edilmesi her servisin en iyi olduğu bölümde çalışıyor olmasını sağlamaktadır. Duran ve akışkan verilerin ayrı ayrı işlenmesi duyulan ihtiyaç, hafıza gereksinimleri ve işlemci yüklerinin de ayrılmasını sağlamaktadır. Koordinatör ve aracı düğümlerin birbirinden ayrılmasıyla, kümelemede

verinin dağıtılması ve sorgu işlemlerinin gerçekleştirilmesi konularında çalışan iki farklı servisin etkin şekilde çalışması sağlanmaktadır. Şekil 3.5’de sorgulama ve veri ithal etme süreçlerinin gerçekleşmesi gösterilmektedir.



Şekil 3.4. Druid sorgu ve veri ithal işlemleri [46]

Şekil 3.6’da ise sistemin yönetim katmanı ve dış dünya bağımlılıkları, belirli düğümlerin birbirleri ile olan bağlantıları ve bağımlılıkları gösterilmektedir.



Şekil 3.5. Druid yönetim katmanı [46]

Bu kısma kadar Druid’in normal çalışması ve kullanımı açıklanmıştır. Ancak Lamda Mimari için kullanımı bu sıradan çalışma şeklinden farklıdır. Bu nedenle, Druid’in

sağlamış olduğu ara yüzler kullanılarak, kaynak kodları değiştirilmiş ve akışkan veri ve duran veri işleme özellikleri Lamda Mimari prensiplerini destekleyecek şekle getirilmiştir. Akışkan veri düğümlerinde donanımın yani belleğin sağlayabildiği ölçüde gerçek zamanlı veriler hafızada tutulmaktadır. Duran veri katmanının belirli zamanlarda çalışarak daha önceki çalışma zamanından, o ana kadar olan tüm verileri tekrar işleyip sunum katmanını güncellemesi ve aynı zamanda AVK'daki işlediği kadar veriyi hafızadan silmesi gerekmektedir. Bu işlemin olabildiğince hızlı şekilde tamamlanması gerekmektedir. Bu nedenle Druid kodlarında bir takım değişiklikler yapılarak kaynak kodları yeniden derlenmiş ve Lamda Mimariye uygun hale getirilmiştir. Normal şartlarda sistemin AVK verilerini silerek, DVK verilerini kullanıma sunabilmesi için farklı komutlar gönderilmesi ve veri dizinlemesi ile metaveri'lerin ZooKeeper üzerinden koordine edilmesi gerekmektedir. Bu işlemin bir defada ve bölütlerin yapısı bozulmadan tamamlanabilmesi sağlanmıştır.

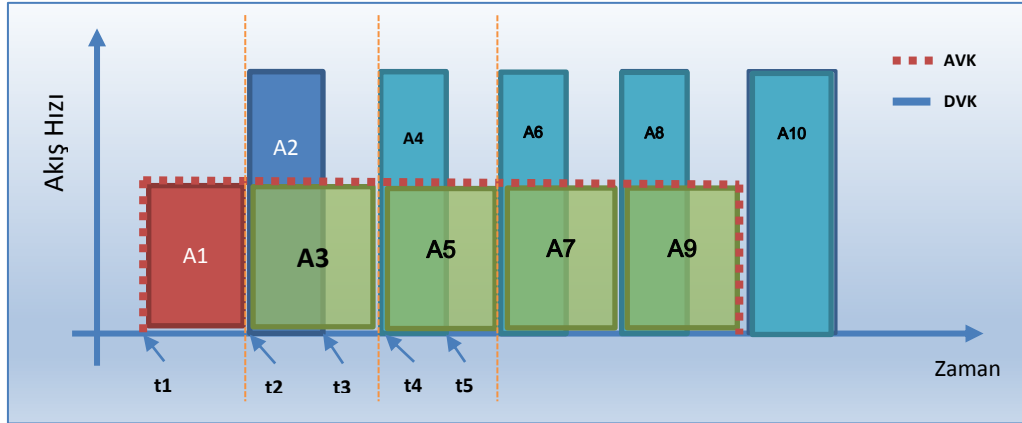
Geliştirilen Lamda Mimaride AVK katmanının olabildiğince düzenli aralıklarda, en az düzeyde sistem üzerine yük getirecek şekilde çalışan bir izleme sistemi de ayrıca geliştirilmiştir. Çalışma anında AVK ve DVK katmanları izlenerek YKK tarafından gerek görüldüğü zamanlarda DVK'nın çalıştırılması sağlanmıştır. Sistem durumu, Druid fonksiyonel olarak kümeleme yapısında çalışan tüm servisleri üzerinde REST API uygulama programlama ara yüzü kullanılarak, yapılandırma dosyasında izlenecek metriklerin etkin hale getirilmesiyle takip edilmektedir. Bazı sürekli izlenmesi gereken hassas metrikler ise ZooKeeper üzerinden de takip edilmektedir.

Bir Büyük Veri işleme sistemi, farklı konumlardaki kaynaklardan beslendiğinde akışkan verilerin veri yoluna ulaşma süreleri de farklılık gösterebilmektedir. Aynı zamanda verinin tam olarak bir defa ya da en az bir defa ulaşması durumları sıklıkla gözlemlenmektedir. Karşılaşılan en kötü durumlardan birisi olan verinin gecikerek gelmesi, zaman penceresine göre işlem yapan SK'nın o zaman penceresi için işlemlerini tamamladıktan sonra geciken veriyi göz ardı etmesine ve işleyememesine neden olmaktadır. Daha önce de belirtildiği üzere bu gibi hatalar DVK tarafından düzeltilmektedir. Druid geciken veri ile karşılaştığında kendi üzerinde bulunan kurallara göre veriyi işleme almamaktadır. Bu durum, akışkan veri düğümlerindeki yapılandırma dosyası içerisinde, gecikerek gelen verinin ne kadar zaman sonra gelirse işlenmeyeceğinin ya da geciken verinin hiç işlenmeyeceğinin belirtilmesi ile ayarlanabilmektedir. Bu çalışmada Lamda Mimarinin

geciken verileri ne kadar zaman içerisinde ve ne şekilde düzelttebildiğinin net olarak takip edilebilmesi için, veri tabanının 2 dakikadan fazla geciken verileri işlememesi yapılandırma dosyasında belirtilerek ayarlanmıştır.

3.3.5. Yönetim/Koordinasyon katmanı

Yönetim/Koordinasyon katmanı Lamda Mimarinin en önemli katmanlarından birisidir. Temelde, sistemde gerçekleşen olayları, AVK ve DVK'nın veri işleme esnasında hafıza ve CPU yüklerini takip ederek, gerekli zamanlarda DVK'nın çalışmasını sağlamak, veri ithal işlemlerinin sonuçlarını kontrol etmektedir. Ayrıca Lamda Mimarinin en önemli özelliklerinden birisi olan değişmeyen (immutable) veri setleri üzerinden yeni analizler yapılması ve eski analiz sonuçlarının ya da kullanılan algoritmaların değişmesi nedeniyle yeni analiz gerekmesi durumunda DVG ve AVG'lerin oluşturulmasını da yönetmektedir. Sistemin çalışmaya başlaması ile birlikte veri yolundan gelen verilerin akış hızlarının ölçülmesi ile elde edilen değerlere göre sistemin azami veri işleme yetenekleri çerçevesinde DVK'nın, AVK üzerinde fazla yük oluşmadan çalıştırılması gerekmektedir. Bu şekilde DVK tarafından işlenen verilerin miktarı kadar veri, AVK'nın kaynaklarını baskı altına almamak ve güvenilirliği etkilememek için AVK'nın kaynaklarından silinerek, gerekli analiz sonuçları derin saklama alanına iletilir. Temelde veri yolundan gelen veri akışının hızına, AVK üzerindeki kaynak tüketimine ve DVK'nın verileri işleme yeteneğine bağlı olarak bu işlem gerçekleştirilir. Ancak tasarım zamanında verilmesi gereken bu kararın, uygulama alanına göre seçilmesi gerekmektedir. DVK'nın çalışmasının başlatılması için periyodik olarak çalıştırma yöntemi seçilebileceği gibi, kaynak tüketimi göz önüne alarak gerekli sıklıklar ile çalıştırma yaklaşımı da benimsenebilir. Her iki yaklaşımın kullanılması da mümkün olsa da, bir Büyük Veri sisteminde kaynak planlamasının göz önüne alınması gerekmektedir. Kümeleme kaynaklarının en iyi şekilde kullanımı çok büyük önem arz etmektedir. Bu çalışmada periyodik olarak veri işlenmesi yerine kümeleme kaynakları göz önüne alınarak DVK'nın başlatılması yaklaşımı benimsenmiştir. Şekil 3.7'de örnek bir veri işleme için akış hızı – zaman grafiği gösterilmiştir.



Şekil 3.6. AVK ve DVK için örnek bir veri işleme akış hızı - zaman grafiği

Sistemin birbiri ile koordineli olarak çalışması Şekil 3.7’de gösterilmiştir. Çalışma prensibinin açıklanabilmesi için veri yolundan gelen verilerin akış hızının tüm analiz süresince değişmediği varsayılmaktadır. Sistemin AVK ve DVK üzerinde hiç veri bulunmadığı bir durumdan (cold start) başlayarak çalışması aşağıdaki gibi özetlenebilir:

$t=t_1$ anında sistem üzerinde bir veri bulunmamaktadır. Bu anda YKK, veri yolu katmanını ve AVK’yı aktif hale getirir ve gerçek zamanlı veri işlenmesine başlanmış olur. $t=[t_1, t_2]$ zaman aralığında sadece akışkan veri katmanında veriler işlenerek SK’ya gönderilir. Bu süre içerisinde gelen verilerin birikmesi ile AVK’da bir miktar kaynak tüketimi olur. $t=[t_2, t_3]$ zaman aralığında, AVK’daki kaynak tüketimi belirli bir seviyeye geldiğinde, YKK tarafından DVK çalıştırılır. DVK çalışma komutunu aldığı anda veri yolundan gelmiş ve duran hale dönüştürülmüş veriler üzerinde analizlerini olabildiğince hızlı olarak gerçekleştirir. $[t_2, t_3]$ zaman aralığında daha önceki zaman aralığı olan $[t_1, t_2]$ de gelen verilerin hepsinin işlenmesi sağlanır. İşlenecek veri kalmayınca DVK çalışmayı durdurur ve işlediği tüm bilgiyi SK’ya iletir. Akış hızı – zaman grafiğinin altındaki alan belirli bir zaman aralığında işlenmiş olan veri miktarını vermektedir. Bu bağlamda incelendiğinde AVK’nın işlediği veri miktarı A1, DVK’nın işlemesi gereken veri miktarı A2 olarak hesaplanabilmektedir. DVK’nın veri işlemede geride kalmaması için $[t_2, t_3]$ zaman aralığında A1 kadar veri işlenmesi yani $A_1=A_2$ olması gerekmektedir. Ancak $A_1 > A_2$ olduğunda AVK’dan gelen verilerin yeterli zaman içerisinde işlenememesinden ötürü DVK geride kalmış durumdadır. Bu durumda yeni gelen veriler ile birlikte DVK’nın işlemesi gereken yük daha da artmaktadır. Bir sonraki periyotta bu durum aynen tekrarlanmaktadır. En sonunda DVK veri işlerken AVK’nın oldukça gerisinde kalır ve bu durumda Lamda Mimarinin güvenilir veri işleme ve DVK’nın, AVK’da oluşabilecek

hataları-eksiklikleri gidermesi gecikmektedir. Bu durumun oluşmasına hiçbir zaman müsaade edilmemesi gerekmektedir. YKK'da çalışacak algoritmanın ya da en basitinden tasarımın $A1 < A2$ eşitsizliğini sağlaması gerekmektedir.

$t=[t2,t3]$ zaman aralığında DVK verileri işlerken bile bir yandan da yeni veriler veri yolu üzerinden AVK'ya ve DVK'ya gelmeye devam etmektedir. $t=[t2,t4]$ zaman aralığı sistemin kaynak tüketiminin en fazla olduğu zaman aralığıdır. Kümeleme üzerinde en fazla kaynak baskısı bu zaman aralığında gerçekleşmekte, AVK veri işlerken aynı zamanda da DVK veri işlemeye başladığı için kümelemenin kaynaklarını AVK ile aynı zamanda tüketmektedir. Veri yolundan gelen ve AVK'da işlenmiş bu zaman aralığındaki veriler ancak $t=[t4,t5]$ zaman aralığında DVK tarafından işlenebilmektedir. Bu süre zarfında AVK'nın ve SK'nın gerçek zamanlı olarak gelen tüm veriyi işleyerek kullanıma sunabilmesi gerekmektedir. Bu süre zarfında gelen veri miktarı akış-zaman grafiğinin altında kalan alan olarak hesaplanabilir. Dolayısı ile $A3 = A4$ olduğunda sunum katmanında veriler düzgün olarak sorgulanabilecektir. Eğer $A3 > A4$ şeklinde bir durum meydana gelirse, bu defa yine DVK verileri işlemede AVK'nın gerisinde kalacaktır. Zaman içerisinde gelen diğer veriler ile bu katman daha da geride kalacak ve AVK'da oluşabilecek sorunlardan ötürü hataların giderilmesini çok geç sağlayabilecektir. Bu kesinlikle istenmeyen bir durumdur. Dolayısı ile güvenli olarak çalışması için DVK'daki işlenecek veri miktarının $A3 \leq A4$ koşulunu sağlaması gerekmektedir. Güvenli çalışma ancak ve ancak $A3 \leq A4$ olduğu durumda mümkün olabilecektir.

Gerçek zamanlı çalışacak bir sistem olan Lamda Mimaride, veri yolu üzerinden kendisine gelen verilerin akış hızı zamanla farklılık gösterebilecektir. Veri akışının azalması durumunda AVK'nın işleyeceği veri ve tüketeceği kaynak miktarında azalma olacak ve ($A3$ alanı azalır) bu nedenle DVK'nın işleyeceği veri miktarı da düşecektir ($A4$ alanı azalır). Tam aksi bir durumda AVK'nın işleyeceği veri miktarı artacak ($A3$ alanı artar), ve DVK'nın işleyeceği veri miktarı da artacaktır ($A4$ alanı artar). Tasarım gerçekleştirilirken her çalışma anında ve koşulunda güvenli sınırlar içerisinde kalınması gerekmektedir. Aksi halde DVK'nın veri işlemede geride kalması ve yeni gelen veriler ile işlemesi gereken veri miktarının artması durumu gerçekleşerek SK'daki verilerin güncellenme süresi giderek artacak ve sistem bir Lamda Mimari gibi çalışamayacaktır. Ama yine de verilerin gecikerek işlenmesi ile sistem geç de olsa güvenilir hale gelmektedir.

Anlatılan sebeplerden ötürü YKK'nın veri yolundan gelen verilerin akış hızını da takip etmesi gerekmektedir. Gelen veri hızı ile AVK'nın işleyebildiği veri miktarı arasındaki korelasyon ve kümelemenin (cluster'ın) AVK bölümündeki sistem kaynaklarının azalması ile DVK'nın çalışmaya başlaması kontrol edilmektedir.

Sistemin çalışması sırasında meydana gelecek hatalarda, gerektiğinde, veri yolunun hızı uyarlamalı (adaptive) olarak azaltılarak AVK'nın veri işleme hızı da azaltılır. Bu sırada sorun yaşanan DVK iş parçacığı tekrar başlatılır. Eğer sorun düzelmişse veri yolu hızı normale döndürülerek normal veri akış hızı sağlanır. Aksi bir durumda ise arka arkaya üç defa başarısız olunması durumunda Lamda Mimarinin veri işlemesi tamamen durdurulur. Bu durumda, veri yolu, verileri depolamaya devam eder. Sistemdeki sorun giderildikten sonra YKK kullanılarak en son kalınan zaman aralığından itibaren tekrar veri işlenecek şekilde Lamda Mimari yeniden başlatılır. Bu durum Lamda Mimarinin en kullanışlı özelliklerinden birisidir. Veriler en ham haliyle saklandığından ötürü gerektiği durumlarda önceki bir zaman aralığına dönülerek verilerin işlenmesi mümkün olmaktadır. Bu şekilde sistemde veri kaybı yaşanmaz ve eninde sonunda güvenilir (Eventually Accurate) hale gelinmiş olur. Bu durumun beraberinde getirdiği tek sorun veri depolama ihtiyacının artmasıdır. Ancak günümüzde diskler oldukça ucuzladığı için güvenilirliğin ve veri kaybının yaşanmaması adına kabul edilebilir bir durum olduğu değerlendirilmektedir.

3.3.6. Gerçek zamanlı uygulama

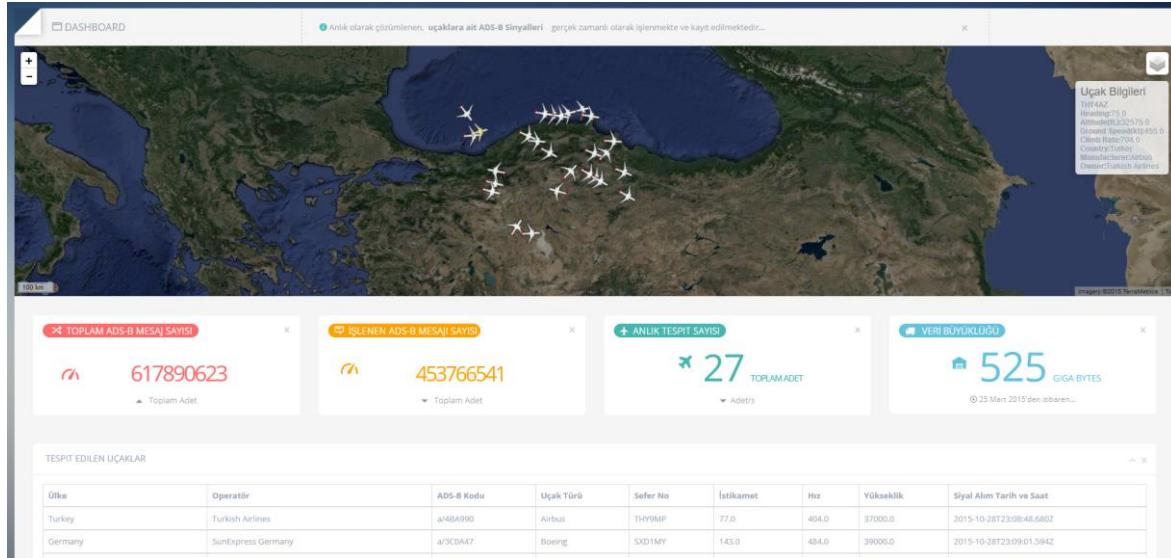
Bu çalışmada klasik Lamda Mimaride olmayan gerçek zamanlı uygulama, AVK'da işlenmekte olan verilerden faydalanılarak bir başka ek özellik olarak Lamda Mimariye kazandırılmıştır. Lamda Mimaride verilerin zaman damgalarına göre SK'dan sorgulanması belirli bir zaman çözünürlüğüne göre gerçekleştirilmektedir. Gerçek zamanlı veri AVK tarafından işlense de sadece analitik hesaplamalar yapılarak sonuçlar SK'ya gönderilir. Canlı olarak sorgu yapılması için gelen verilerin mili saniye çözünürlüğünde dizinlenmesi gerekmektedir. Bu işlemin pratikte yapılabilmesi mümkün olsa da bir Büyük Veri sistemini bile zorlayabilecek bir özelliğe sahiptir ve güçlü donanımlar gerektirmektedir.

Gerçek zamanlı uygulamada AVK'dan gelen veriler canlı olarak görselleştirilmektedir. AVK'dan gelen veriler Spark Streaming teknolojisi ile işlenirken, bir kopyası da Redis veri tabanına yazılmaktadır. Bu veri tabanının kullanılmasının nedeni, gelen verilerin

tamponlanarak veri iletiminde sorun yaşanmasını engellemek ve AVK ve DVK'dan farklı ETL işlemlerini de gerçekleştirebilmektir. Redis veri tabanına yazılan veriler Node.js teknolojisi kullanılarak bir web uygulamasına dönüştürülmüştür. Web uygulaması üzerinden WebSocket teknolojisi kullanılarak tüm istemcilere gecikme asgari düzeyde olacak şekilde gönderilebilmektedir. Hem Redis hem de Node.js teknolojisi yatay olarak ölçeklenebildiği için uygulama oldukça fazla sayıda istemci tarafından kullanılabilir. Gerekğinde donanım sayısında artırım yapılarak daha fazla istemci tarafından kullanılması da sağlanabilmektedir. Verinin veri yolu üzerinden iletilmesinden başlayarak istemciye iletilmesine kadar ki gecikme süresinin azami 1 saniye olduğu tespit edilmiştir.

Uygulamada ayrıca, daha önceden tespit edilmiş hava taşıtlarının belirli bir zamandan itibaren konum değişimlerinin tekrar oynatılması (replay) özelliği de bulunmaktadır. Lamda Mimaride bulunan en önemli özelliklerden birisi olan verilerin ham olarak saklanması ile bu işlem gerçekleştirilmektedir. İstemcinin Node.js web uygulaması üzerinden verdiği komut ile bu istek YKK'ya iletilmektedir. YKK istenen zaman aralığındaki bilgileri başka bir Kafka başlığı (topic) altında yayınlamakta ve Spark Streaming ile işleyerek tekrar Redis veri tabanına göndermektedir. Node.js gelen verileri yeniden WebSocket ile istemcilere dağıtmaktadır.

Resim 3.1'de, gelen verilerin canlı olarak gösterilmesini sağlayan uygulama arayüzü gösterilmektedir. Bu ara yüz vasıtası ile hava taşıtlarının koordinatları, yükseklikleri, yönleri, ait oldukları ülke, operatörleri, model ve marka bilgileri canlı olarak kullanıcıya gösterilmektedir.



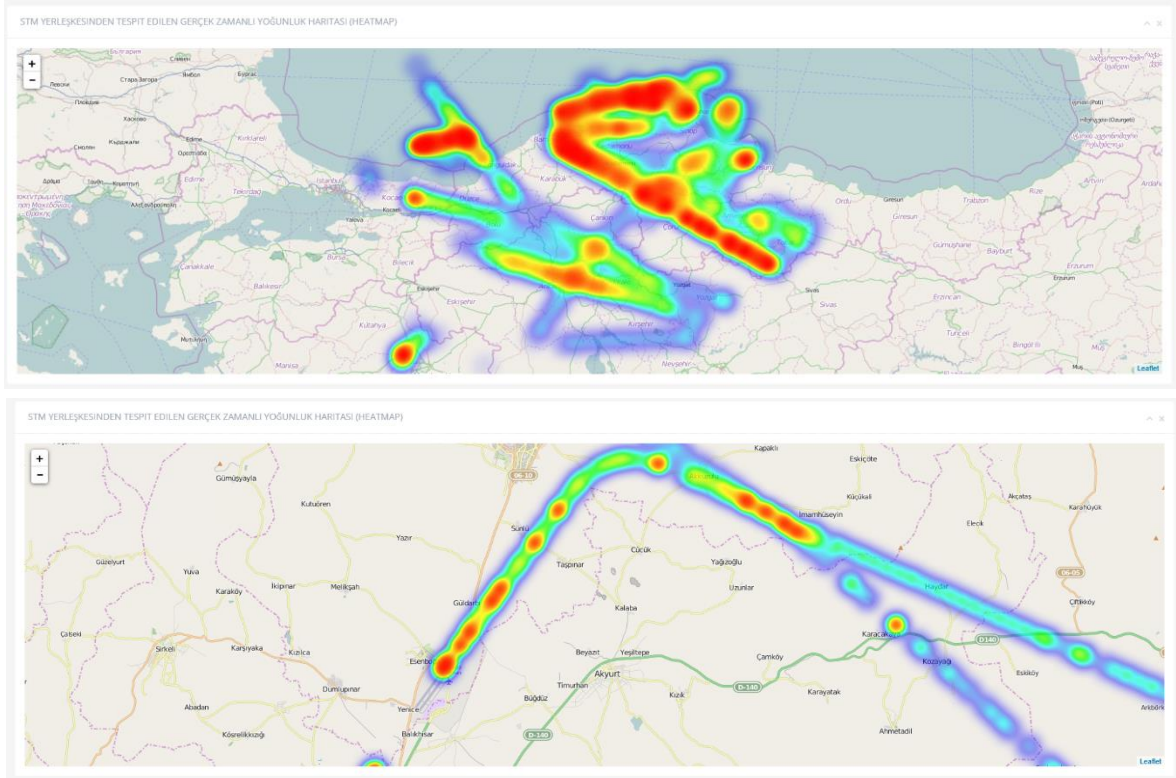
Resim 3.1. Canlı veri gösterim ara yüzü

Resim 3.2’de, gelen verilere ETL işlemi uygulandıktan sonra elde edilen bilgilerden üretilen kelime bulutu görseli sunulmuştur. Bu gösterimde hava taşıtlarının ait olduğu ülkeler, hava taşıtlarının operatörleri ve hava taşıtı türleri belirlenmektedir. Havadaki taşıt sayısına göre anlık olarak bu veriler güncellenerek metin büyüklükleri ayarlanmaktadır. Hava taşıtı fazla olan ülke, operatör ve uçak tipleri daha büyük, az olanlar ise daha küçük boyutlu metinler olarak gösterilmektedir. Bu şekilde, havadaki durumun bir resmi çekilerek farklı açılardan verilerin görselleştirilmesi sağlanmaktadır.



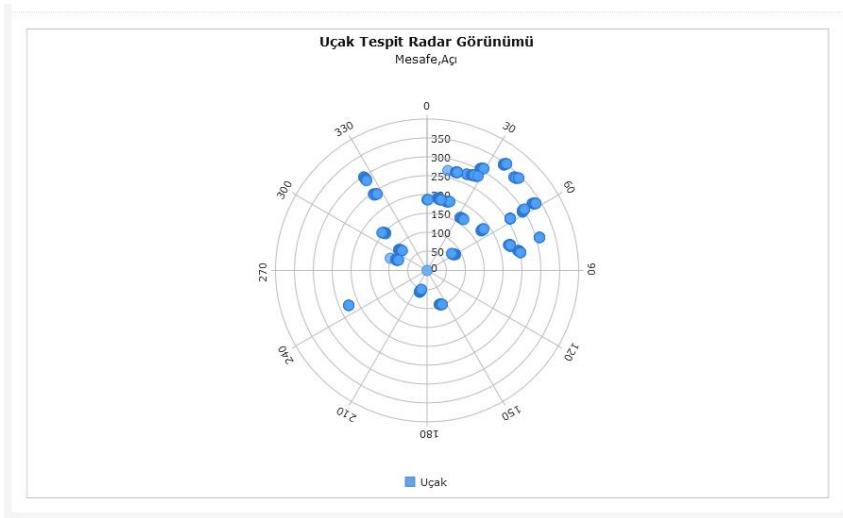
Resim 3.2. Kelime bulutu gösterimi

Resim 3.3’de uçakların en çok geçtiği bölgeler ısı haritasına (heatmap) dönüştürülerek verilmiştir. Bu gösterimde, belirli bir bölgeden yoğun sinyaller alındığında rengi kırmızıya, daha az sayıda ADS-B sinyali tespit edildiğinde ise rengi mavi dönüşen alanlar oluşturulmaktadır. Harita üzerinde yakınlştırılma yapıldığında uçuş güzergâhları ve hava koridorları net bir şekilde belirlenebilmektedir. Resim 3.3’de ayrıca Esenboğa hava limanından kalkan uçakların yapmış olduğu manevralar gösterilmektedir.



Resim 3.3. Canlı ısı haritası (HeatMap) gösterimi

Resim 3.4’de canlı olarak hava taşıtlarının mesafe ve açı bilgileri radar grafiği olarak gösterilmektedir. Gelen veriler AVK’da canlı olarak işlenirken Haversine formülü kullanılarak uçağın mevcut koordinatları ile antenlerin konuşlandırıldığı yer arasındaki mesafe ve açı (yön) bilgileri hesaplanmaktadır. Buradan elde edilen bilgi canlı olarak istemciye gönderilmekte ve burada radar grafiğine dönüştürülmektedir.



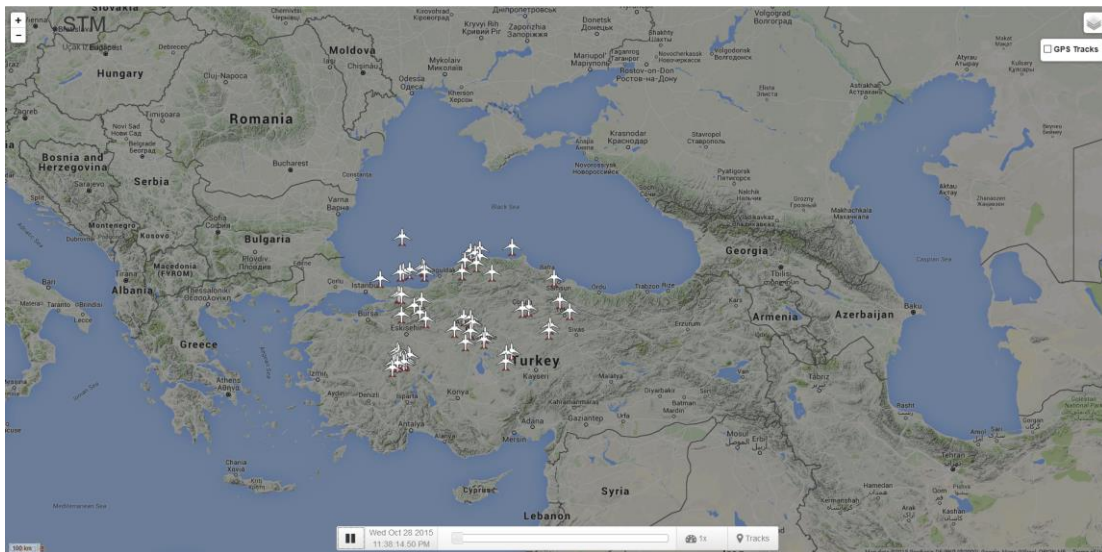
Resim 3.4. Hava taşıtlarının canlı olarak radar grafiği olarak gösterimi

Resim 3.5’de tespit edilen hava taşıtı bilgileri sıralamalı olarak liste görünümünde sunulmuştur. Bu gösterimde havadaki hava taşıtı bilgilerine ve sayılarına göre, hava taşıtlarının ait olduğu ülke, operatör ve türlerine göre gruplanması yapılarak en çok sayıda olandan en aza sayıda olana doğru sıralı bir biçimde gösterilmektedir. Gelen bilgilere göre canlı olarak liste güncellenmektedir. Sıralamada bir değişiklik olduğunda sayısı artarak üst sıralara çıkan ülke, operatör ve tür yeşile dönüştürülerek güncellenmekte, sayısı azalanlar ise kırmızı renge dönüştürülerek alt sıralara indirilmektedir.

== UÇAKLARIN AIT OLDUĞU ÜLKELER		== TESPİT EDİLEN HAVA YOLLARI		== TESPİT EDİLEN UÇAK TÜRLERİ	
Ülke	Uçak Sayısı	Havayolu	Uçak Sayısı	Uçak Türü	Uçak Sayısı
Türkiye	7	Turkish Airlines	6	Airbus	14
United Arab Emirates	4	Emirates Airline	2	Boeing	7
Germany	3	Air Berlin	2	UAC	2
UK	2	UAC	2		
Belgium	1	Ethad Airways	2		
Bermuda	1	SunExpress Germany	1		
Greece	1	Nordwind Airlines	1		
India	1	Aegean Airlines	1		
Oman	1	Zagros Airlines	1		
Netherlands	1	Air India	1		
Azerbaijan	1	Oman Air	1		
		Air Arabia	1		
		KLM Royal Dutch Airlines	1		

Resim 3.5. Hava taşıtı canlı liste gösterimi

Resim 3.6’da hava taşıtı bilgilerinin yeniden oynatılması özelliği gösterilmiştir. Canlı olarak gelen veriler veri yolu üzerinden DVK’ya ham olarak kayıt edilir. Gerekğinde bu veriler tekrar hareketli hale dönüştürülerek AVK’ya iletilir ve buradan da gerçek zamanlı uygulama tarafından gösterilir. Bu görselleştirmede, veri yolu aktarım hızı artırılarak duran veriler normalden 30 kat daha hızlı olarak gönderilebilmekte ve görsel üzerindeki hava taşıtlarının daha hızlı izlenmesi sağlanabilmektedir.



Resim 3.6. Hava taşıtı bilgilerin yeniden oynatılması (replay)

4. LAMDA MİMARİ GELİŞTİRİLMESİ VE TEST EDİLMESİ

Bu bölümde, bir önceki bölümde tasarlanmış olan LM'nin gerçekleştirilmesi ve toplanan veriler ile test edilmesi sonucu elde edilen bulgular paylaşılmıştır. Daha önceki bölümlerde detaylı olarak açıklanan teknolojiler yardımıyla veri toplama ve iletim yöntemleri kullanılmış, veri işleme algoritmaları ile gerçek zamanlı ve büyük hacimlerde gelen verilerin, LM üzerinde işlenerek canlı olarak sorgulanması sağlanmıştır. Öncelikle her bir veri işleme katmanı ayrı ayrı geliştirilmiş ve aynı veri yolu üzerinde birlikte çalışabilecek şekilde bütünleştirilmiştir. Sonuçların analizinde ve incelenmesinde Lamda Mimarilerin eninde sonunda güvenilir (eventually accurate) olarak çalıştığı gerçeği dikkatte alınmıştır. Sistem analizi ve veri işleme operasyonları yürütülürken gerçek hayatta karşılaşılan durumlar bire bir olarak denenmiş ve performans test edilmiştir. Zaman odaklı bir işlem yapıldığı için öncelikle kullanılan tüm donanım alt yapısındaki bilgisayar saatlerinin NTP sunucu ve protokolü ile kusursuz şekilde eşzamanlı hale getirilmesi sağlanmıştır.

Bu bağlamda öncelikle sistemin veri işleme kapasitesinin azami hangi düzeyde olduğunun tespit edilmesi önemlidir. Çalışmanın ilk kısmında sistemin azami yük altındaki performansı ve davranışı incelenmiştir. Daha sonra normal çalışma durumundaki veri işleme yeteneği dikkatlice araştırılmıştır. Diğer bir testte ise gerçek dünyada en çok karşılaşılan verinin gecikmesi ve aynı verinin tekrar tekrar gelebilmesi durumlarında sistemin davranışı gözlemlenmiştir. Yine gerçek hayatta karşılaşılabilecek bir durum olan DVK'nın bir nedenden ötürü çalışmaması durumu ve bu durumda tekrarlı veri gelmesi, veri gecikmesi koşullarında sistem performansı test edilmiştir. Sistemin gerçek zamanlı gelen verileri işleyen katmanı olan AVK'nın bir nedenden dolayı çalışmaması durumunun görselleştirilmesi teknik olarak incelemeye değer bulunmamıştır. Bu nedenle bu çalışmaya dâhil edilmemiştir. Bunun başlıca sebebi tasarımda, AVK üzerindeki veri yükünün yönetim/koordinasyon katmanı tarafından sürekli olarak gözlemlenmesi ve yükün belirli bir düzeye geldiğinde DVK'nın çalışmaya başlaması kararının alınmasıdır. AVK çalışmadığı durumda, DVK'nın AVK'nın ne kadar yüklendiği bilgisini alamayacağı ve sürekli olarak yüksüz olduğu belirleneceği için DVK çalışmaya başlamayacağından ötürü bu inceleme gerçekleştirilen tasarım için ne yazık ki mümkün değildir.

AVK ve DVK'nın bir birinden bağımsız olarak çalıştırılacağı bir tasarımın da gerçekleştirilmesi mümkündür. Ancak bu tasarım deterministik ve güvenilir bir çalışma olmayacaktır. Sisteme gelen veri miktarının artmasıyla birlikte, DVK'nın sürekli olarak belirli zamanlarda (aynı periyotla) çalışması (örnek: her 10dk'da bir) güvenli bir çalışma sağlamamaktadır. Gelen veri miktarının artmasıyla, DVK'nın tasarımında karar verilen zaman aralığında veriyi işleyerek kendisine ait DVG'leri oluşturması ve AVK'daki işlediği kadar veriyi yok ederek AVK'nın üzerindeki yükü ortadan kaldırması/azaltması asla mümkün olmayacağı durumlar gerçekleşebilecektir. Bu durumda DVK'nın asla verileri zamanında işleyip kendisi ile ilgili DVG'leri oluşturamayacağı öngörülmüştür. Böyle bir durumla gerçek hayatta neredeyse hiç karşılaşılmaktadır. Bu nedenle tasarımda AVK'nın sürekli olarak gözlemlenmesi için bir alt yapı kurulmuş ve AVK üzerinde yük arttığında DVK'nın veri işlemede geri kalmaması için mevcut tasarımda belirli bir güvenlik payı bırakılarak gelen verinin işlenmesi sağlanmıştır. Literatür taramasında da görüldüğü üzere, yapılan çalışmalar ve analizlerin çoğu, gerçek büyük verilerle test edilmemiştir. Neredeyse hiç birisi sunum katmanı üzerine bir inceleme gerçekleştirilmemiştir. Bir çoğu sadece DVK üzerine yoğunlaşmıştır. Bazı çalışmalarda ise LM'de AVK olduğu halde, AVK'nın üzerine asgari yük bindirilmesi amaçlanarak bazı algoritmalar geliştirilmiş ve test edilmiştir. Ancak bu çalışmalarda sistemin sunum katmanı LM yaklaşımına uygun olarak geliştirilmediği ve sadece DVK veri işlemesi kullanılmadığı için sorgu sonuçlarının doğruluğu oldukça tartışmalıdır. Bu durumda sonuçlar sadece duran veri analizi yapılarak elde edilmekte ve sistem Lamda Mimariye uygun olarak çalışmamaktadır.

Sistemin güvenilirliğinin ve performansının incelenmesi için ayrıca bir test sistemi geliştirilmiştir. Bir Büyük Veri sistemi test edileceği için, geliştirilen test sisteminin de buna uygun olması, yüksek hacimli ve hızlı veriyi işleyebilmesi ve sonuçları analiz edebilmesi gerekmektedir. Sistemin güvenilirliğinin test edilmesi için bilimsel metriklerin kullanılması şarttır. Gerçekleştirilen araştırmalarda bu tür sistemlerin performansının test edilmesi için en uygun yöntemin Geri Getirme (Recall) ve Kesinlik (Precision) değerlerinin hesaplanması olduğu tespit edilmiştir. [30-31]. Bu iki metrik performans göstergeleri (Key Performance Indicators) olarak aşağıdaki gibi tanımlanmıştır;

- Geri Getirme: Birim ya da belirli bir zaman aralığında (zaman penceresi), bir deneyin sonuçlarından ne kadar tespit edilebildiği bilgisidir. 0 ile 1 arasında sonuç elde edilir. Sık sık yüzde olarak da ifade edilebilmektedir. 0 geri getirmenin başarısız olduğunu, 1

ise verilerin tamamının getirildiğini göstermektedir. Sıfır ile bir arasındaki değerler ise geri getirme için başarı oranını göstermektedir.

- Kesinlik: Geri getirme analizi tamamladıktan sonra, elde edilen sonuçların gerçek veri ile ne kadar aynı olduğu bilgisidir. 0 ile 1 arasında sonuç elde edilir. Sık sık yüzde olarak da ifade edilebilmektedir. 0 kesinliğin başarısız olduğunu, 1 ise geri getirilen verilerin tamamının test setinde olduğunu göstermektedir. Sıfır ile bir arasındaki değerler ise kesinlik için başarı oranını göstermektedir.

Söz konusu bu iki metrik makine öğrenimi algoritmalarının test edilmesi için yaygın şekilde kullanılmaktadır. Analiz sonuçlarının elde edilmesinden sonra sistem performansının test edilmesine oldukça uygun metriklerdir. Bir makine öğrenimi algoritmasının sınıflandırma amacıyla çalışması sonucu elde edilen test veri setleri, kestirim sonucu elde edilen veri setleri ile karşılaştırılarak Geri Getirme ve Kesinlik değerleri hesaplanır. Gerekğinde alıcı operasyon karakteristiği eğrileri (ROC) çizilerek kullanılan makine öğrenimi algoritmasının veri seti üzerindeki sınıflandırma yeteneği ölçülür. Tasarlanan LM içinde, gelen verilerin belirli sorgular sonucunda oluşacak verilerden ne kadarını getirebildiği ve geri getirdiği verilerin ne kadarının doğru olduğunun belirlenmesi gerekmektedir. Bu durumda sistemi bir makine öğrenimi algoritması gibi düşünerek sonuçların değerlendirilmesi en doğru yöntemlerden birisi olacaktır ve bu yöntemin literatürde kullanıldığı da belirlenmiştir [30-31].

Havadan canlı olarak toplanan sinyaller, gerçek zamanlı olarak çözümlenerek işlenecek duran veri haline dönüştürülmüştür. Daha sonra bu veriler çalışmadaki LM tasarımında kullanılan teknolojiden başka bir teknoloji olan Apache Hadoop ve Hive ile işlenerek test verileri (Ground Truth) elde edilmiştir. Test verileri, veri yolu katmanına geliş zamanına göre zaman pencerelerine ayrılarak hesaplanmıştır. Bu şekilde gelen verilerin zamanına göre, tüm zaman pencerelerinde sorgulamalar gerçekleştirilebilmiştir. Sistem performansının test edilmesi test/doğrulama altyapısı kullanılarak yapılmıştır. Bu altyapıda daha önce de belirtildiği gibi Apache Hadoop ve Hive teknolojileri kullanılmaktadır. Bunun başlıca nedeni, LM’de kullanılan duran veri işleme teknolojisinden farklı bir teknoloji kullanılarak, bağımsız bir test ortamı oluşturulmak istenmesidir. Verilerin hem test hem de gerçek çalışma ortamında Apache Spark kullanılarak doğrulanması teknik olarak mümkün olsa da etik açıdan uygun bulunmamıştır. Bu nedenle test sistemi

MapReduce tabanlı Apache Hadoop üzerinde Hive kullanılarak geliştirilmiş ve test verileri elde edilmiştir.

Büyük Veri uygulamalarının en gelişmiş örneği ve kullanımı olan LM'nin geliştirilerek test edilmesi için buna uygun bir donanım kullanılması gerekmektedir. Sunucular üzerine binecek yük oldukça fazla olduğu için buna uygun ve gelişmiş bir donanım üzerinde geliştirme ve test faaliyetleri yürütülmüştür. Donanımın çalışma sırasında arızalanmadığından ve her bir testin deterministik olarak yapıldığından emin olmak için, donanım ve yazılımlardan toplanan günlükler gerçek zamanlı olarak Apache Flume tabanlı sistem izleme alt yapısı ile takip edilmiştir. Çizelge 4.1'de bu çalışma için kullanılan Büyük Veri donanım alt yapısı gösterilmiştir:

Çizelge 4.1. Lamda Mimari donanım altyapısı

Marka/Model	Tip	Adet	İşlemci x Core	Bellek (GB)	Depolama
Dell/PowerEdge R730	Sunucu1	12	2 x 14 core@2.2 GHz (E5-2683)	384	8 x 6 TB
Dell/PowerEdge R320	Sunucu2	13	1 x 8 core @ 1.9 GHz (Xeon E5-2440)	128	4 x 4 TB
Dell/OptiPlex 9020	Masaüstü	20	1 x4 core @ 3.6 GHz (i7-4790)	32	1 x 2 TB

Donanım olarak temelde iki farklı sunucu tipi ve masaüstü bilgisayarlar kullanılmıştır. LM için kullanılan donanımın toplam belleği 6144 TB olup, toplam disk kapasitesi 784 TB'dır. Donanım üzerinde aynı anda çalıştırılabilecek iş parçacığı (thread) adedi ise 880'dir. Sunucu1 tipindeki sunucular hesaplama işlemleri için kullanılmıştır. Sunucu2 tipindeki sunucular ise, Büyük Veri altyapısında veri ithali (data ingestion), veri yolu (Data Bus) altyapıları ve sunucu1 tipindeki sunucuların kaynak yönetim servislerinin (YARN) çalıştırılması için kullanılmıştır. Masaüstü bilgisayarlarda ise sistem izleme altyapısı kurulmuş, ayrıca yazılım geliştirme faaliyetleri gerçekleştirilmiştir. Donanımların üzerine Linux Ubuntu Server'ın 14.04 sürümü işletim sistemi olarak kurulmuştur. Tüm sunucular ve masaüstü bilgisayarlar arasındaki ağ haberleşmesi 10 Gigabit'lik fiber optik 3 adet anahtar ile sağlanmaktadır. Apache Ambari yazılımı vasıtasıyla sunucular üzerine Hadoop YARN, Apache Spark, Apache Kafka, Apache Storm, Apache Flume, Apache HBase, Apache Sqoop kurulmuştur. Ayrıca Ganglia, Elastic Search, Kibana, Redis ve Node.js yazılımları da manuel olarak kurulup konfigüre edilmiştir. Kurulan tüm yazılımların öncelikle kendi başlarına sorunsuz olarak çalışması test edilmiştir. Bir birleri ile etkileşerek çalışan tüm yazılımların bağlantıları da ayrıca test edilmiştir. Veri kaybı olmaması ve bir birleri ile etkileşen yazılımların düzgün şekilde çalışmaya başlayabilmesi için, tüm

sistemin belirli bir sıra ile başlatılması ve yine belirli bir sıra ile durdurulması gerekmektedir. Bu gereklilikten ötürü, bu tür işlemleri eksiksiz yapacak yazılım parçacıkları geliştirilmiş ve kullanılmıştır.

4.1. Veri Temini

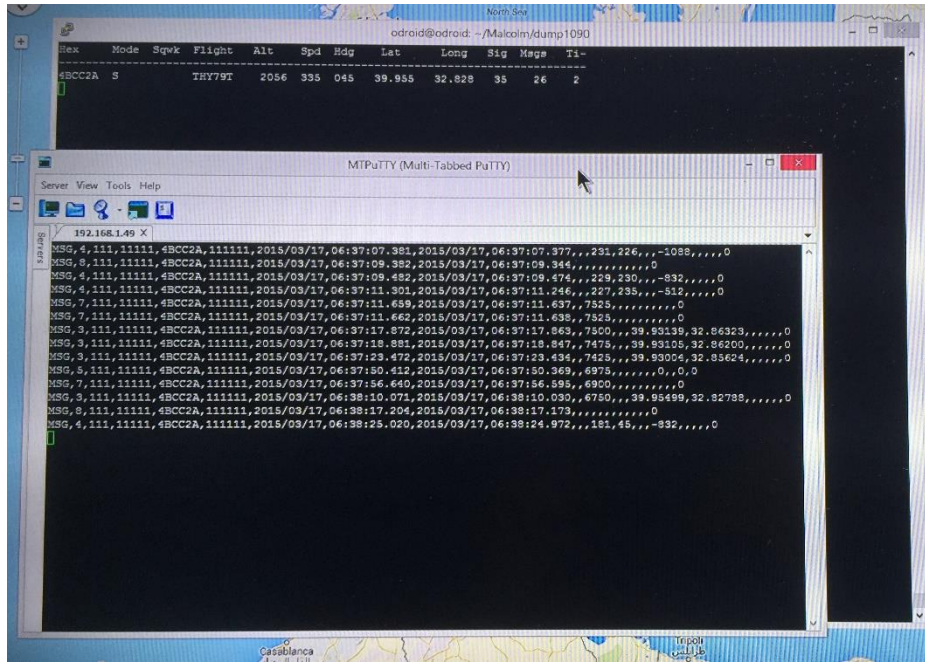
LM olarak tasarlanan ve geliştirilen sistemin test edilebilmesi için gerçek zamanlı, hızlı ve büyük miktarlarda veriye ihtiyaç duyulmaktadır. Bu tip bir verinin elde edilmesi için yazılım tabanlı telsizler (SDR) kullanılarak hava taşıtlarından yayınlanan ADS-B sinyalleri kullanılmıştır. Üç adet diyapazon anten, USB yazılım tabanlı telsiz ve mini bilgisayar vasıtasıyla yayınlanan ADS-B sinyalleri çözümlenerek uçakların yayınladığı gerçek zamanlı bilgiler elde edilmiş ve tasarlanan veri yolu katmanı üzerinden LM'ye yine gerçek zamanlı olarak iletilmiştir. Söz konusu veri toplama sistemi, Bilkent Cyberpark'ta kurulmuştur. Resim 4.1'de veri toplama sisteminin bina içi yerleşimi, Resim 4.2'de ise bina dışı kurulumu gösterilmektedir. Resim 4.3'de veri temin sisteminin çalışması sonucunda elde edilen gerçek zamanlı verilerin en ham hali verilmiştir.



Resim 4.1. Veri toplama sistemi bina içi yerleşimi



Resim 4.2. Veri toplama sistemi bina dışı kurulumu



Resim 4.3. Yazılım Tabanlı Telsizlerden elde edilen ham veri

Resim 4.3’de, yayınlanan ADS-B mesajlarının en ham hali görülmektedir. Burada, sadece tek antenden elde edilen sinyallerin çözülmesi sonucu elde edilen veriler gösterilmiştir. Veri temin sistemi çalıştırıldığında benzer şekilde üç farklı antenden veri elde edilmektedir. Bu veriler veri yolu altyapısına TCP protokolü üzerinden veri soketi (TCP Socket) ile gönderilmektedir. Bu haliyle sistem üç farklı kaynaktan gelen sinyalleri işlemek ve büyük hacimli ve hızlı gelen veriler ile baş etmek zorundadır. Her antenin fiziksel konumu farklı olduğu için, sinyallerin bu antenlere ulaşması ve çözülmesi için

geçen sürelerin farklı olması, gelen verilerin zaman düzleminde senkronize edilmesini zorunlu kılmaktadır. ADS-B sinyallerinin farklı mesaj tiplerine sahip olması ve bu mesaj tiplerinin her birinin çok sık ve tekrarlı olarak gelmesi, veri işlemede zaman senkronizasyonunu zorunlu kılan bir diğer etkidir. Bu nedenden ötürü, sistemin aşırı miktarda ve çok hızlı biçimde gelen akışkan verileri gerçek zamanlı olarak işlemesi, veri hacmi ile baş edebilecek şekilde tasarlanması ve bu zor mühendislik probleminin çözülmesi gerekmektedir. Bu konuda detaylı bilgi ilerleyen alt başlıklar altında verilmiştir.

4.1.1. ADS-B sinyalleri

Hava trafik kontrolü operasyonları, hava trafiğini takip etmek için temel olarak, birincil izleme radarlarını kullanmaktadır. Radar, etraftaki hava taşıtlarına radar darbe sinyalleri göndererek dönüt sinyalleri toplamaktadır. Bu tarz bir operasyonda hava taşıtının kimliği ve yüksekliği belirlenemediği için kısıtlı bir bilgi sayılmaktadır. Bu nedenle hava taşıtına darbe sinyalleri göndererek iletişim kuran ikinci bir izleme radarı kullanılmaktadır. İletişim moduna göre hava taşıtı ikincil radara kimliğini, yüksekliğini ve yön bilgisini iletmektedir. ADS-B bu mantıkla çalışan ModeS iletişim modunu kullanarak periyodik olarak bilgileri ileten bir teknik kullanmaktadır. ADS-B otomatik çalışan bir sistem olarak sınıflandırılmaktadır. Bunun nedeni haberleşme esnasında bir sorgulama (interrogation) işlemine gerek duyulmamasıdır. ADS-B Out servis sağlayıcısı yeteneklerini bildiren bir sinyal yayınlamakta ve ADS-B In servisi de ADS-B mesajlarının alınması işlemini gerçekleştirmektedir. Bu şekilde ADS-B In donanımı bulunan bir hava taşıtı çevresindeki diğer hava taşıtlarını görebilmektedir. ADS-B mesajlarındaki mesaj tipi alanı, o mesajın hangi tipte veri taşıdığını belirlemektedir [47].

Hedefin pozisyonunun takip edilebilmesi için gerekli bilgi; ADS-B mesajının içerisinde enlem, boylam, irtifa ve hedefin hızı alanlarında bulunmaktadır. Her bir bilgi farklı bir mesaj tipi ile gönderilmekte, her bir mesaj tipinin farklı gönderim ve güncelleme sıklığı bulunmaktadır. Örnek olarak hava taşıtının pozisyonunu, yüksekliğini ve hızını belirten mesajlar 2 Hz'lik bir gönderim sıklığına sahiptir. ADS-B mesajları temelde 8 farklı tipte oluşturulmaktadır. Bunlardan Tip 3-7 arası mesajlar hedefin izlenmesi için kullanılabilir. Tip 1 mesajlar sadece hava taşıtının tanımlanmasını sağlayan ve kategorisini belirten bilgiyi içermektedir. Tip 2 mesajlar ise yüzey pozisyonunu bildirmektedir. Tip 8 mesajı, ikincil radarın isteğine cevap olarak gönderilen hava taşıtının

adres bilgilerini içermektedir. Bazı mesaj tipleri için ADS-B formatları aşağıda gösterilmiştir [47]:

Tip	Aircraft Hex-ID	Mesaj Üretim Zamanı	İrtifa
MSG,3,496,211,4CA2D6,10057,2008/11/28,14:53:50.594,2008/11/28,14:58:51.153,,3700			
0,,, 51.45735,-1.02826,,,0,0,0,0			
Enlem	Boylam		

Aircraft Hex-ID hava taşıtına ait 6 basamaklı 16 tabanda bir sayıdır. Mesaj üretim zamanı hava taşıtının bulunduğu pozisyon bilgisini zamanla ilişkilendiren bir zaman damgasıdır. İrtifa alanında gerçek irtifa ya da basınç irtifası bilgisi bulunmaktadır. Enlem ve boylam bilgisi ise hava taşıtında bulunan GPS donanımından alınan pozisyon bilgisini içermektedir. ADS-B Tip 4 Mesajı aşağıda gösterilmiştir [47]:

Tip	Aircraft Hex-ID	Mesaj Üretim Zamanı	Yer Hızı
MSG,4,496,469,4CA767,27854,2010/02/19,17:58:13.039,2010/02/19,17:58:13.368,,,288.6			
, 103.2,,, -832,,,,,			
Track	Tırmanma Oranı		

Yer hızı alanı, hava taşıtının yere göre olan göreceli hız bilgisini içermektedir ve birimi kts'dir. Tırmanma oranı alanının birimi feet/minute olup, hava taşıtının tırmanma bilgisini belirtmektedir. Track alanı ise hedefin derece cinsinden rüzgâra göre düzeltilmiş yönünü göstermektedir.

Tip 5 ve Tip 6 mesajlar hava taşıtı tarafından ikincil izleme radarları tarafından gönderilen sinyaller için yayınlanmaktadır. Bu mesajlar hava taşıtının irtifasının alınması için kullanılmaktadır. Hava taşıtı, ikincil radarlar tarafından gönderilen sinyallere cevap vermek için 1090 MHz frekansını kullanmaktadır. Bu iki mesaj tipi de hava taşıtının irtifa bilgisini içermektedir [47].

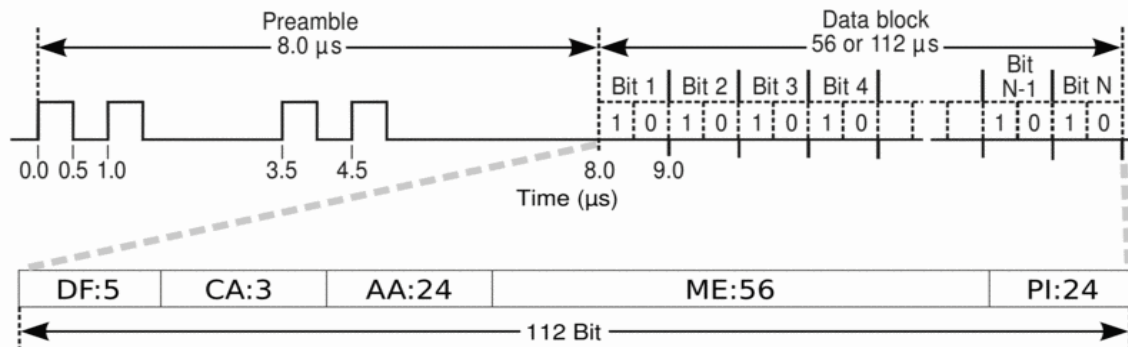
Tip	Aircraft Hex-ID	Mesaj Üretim Zamanı	İrtifa
MSG,5,496,329,394A65,27868,2010/02/19,17:58:12.644,2010/02/19,17:58:13.368,,10000			
,,,,,,0,0,0			

Tip	Aircraft Hex-ID	Mesaj Üretim Zamanı	İrtifa
MSG,6,496,237,4CA215,27864,2010/02/19,17:58:12.846,2010/02/19,17:58:13.368,,33325			
,,,,,,0271,0,0,0,0			

Tip 7 mesaj da aynı şekilde irtifa bilgisini içeren bir mesajdır, ancak bu defa hava taşıtı bu bilgiyi çarpışma önleme sistemi (TCAS) tarafından gönderilen istek sinyaline cevap olarak göndermektedir. Çarpışma önleme sistemi bulunan hava taşıtları birbirlerinin varlığından haberdar olduğunda uyarı sinyali göndermektedir. Bu uyarı sinyali de 1090 MHz frekansından gönderildiği için ADS-B alıcısı tarafından çözülebilmektedir. Normal durumda bu sinyal yayınlanmayacağı için tespit edilememektedir [47].

Type	Aircraft Hex-ID	Mesaj Üretim Zamanı	İrtifa
MSG,7,496,742,51106E,27929,2011/03/06,07:57:36.523,2011/03/06,07:57:37.054,,3775,,			
,,,,,,0			

ADS-B In sinyalinin temelde birkaç kullanım amacı bulunmaktadır. Yerde, yer trafiğinin izlenmesi ve çarpışmaların önünde geçilmesi, havada ise durumsal farkındalık sağlamak için kullanılmaktadır. Şekil 4.1’de ModeS sinyalinin modülasyonu ve genişletilmiş squitter formatı kullanımı gösterilmiştir [48].



Şekil 4.1. ADS-B ModeS sinyali modülasyonu [48]

ADS-B ModeS sinyalinin veri bağı (data link) iki frekansta çalışmaktadır. 1030 MHz frekansı uplink için 1090 MHz frekansı ise downlink haberleşme için kullanılmaktadır. Uplink sinyalleri karadan havaya olan görüşme ve bilgi servisleri için kullanılmakta olup downlink sinyalleri ise uplink sinyallerine verilen cevaplar için havan karaya ya da havadan havaya haberleşme için kullanılmaktadır. ModeS için iki format türü tanımlanmıştır [48]. Bunlardan birincisi 56 bit uzunluğunda olan kısa diğeri ise 112 bit uzunluğunda olan uzun formattır. Uzun format genel belirtilmeyen mesajların gönderimi için ek bilgi alanları içermektedir. ModeS mesajları, darbe pozisyon modülasyonu yöntemi kullanılarak sembol oranı 1 Msembol/s olacak şekilde gönderilmektedir.

Gönderilen her mesaj 8 mikro saniyelik başlangıç sinyali ile başlamaktadır [49]. Başlangıç sinyalinin 56 bitlik ya da 112 bitlik veri bloğu sinyali takip etmektedir. Bu alanda veri; hava taşıtının pozisyonu, irtifası ve hızı gibi bilgileri taşımaktadır. Veri bloğunun ilk 5 biti ADS-B mesajının downlink formatını temsil etmektedir. Bunu takip eden üç bit (CA) ADS-B mesajının 8 alt tipini belirtmektedir. Sonraki 24 bit ise her bir uçağa ait ICAO hava taşıtı adres kodunu göstermekte ve mesaj kaynağını belirtmektedir. Bundan sonraki 56 bitin anlamı CA bitine göre değişmektedir. Mesaj tipine göre hız, pozisyon ya da koordinat olarak yorumlanabilmektedir. Son 24 bit ise eşlik (Parity) bitleri olarak kullanılmaktadır ve ADS-B mesajı alındığında bütünlüğünün kontrolü için kullanılmaktadır [49].

ADS-B sinyalleri 1090 MHz frekansından saniyede ortalama 6.2 adet iletim olacak şekilde gönderilmektedir. Saniyede 2 defa pozisyon, 2 defa hız ya da koordinat mesajları gönderilirken, saniyede 10 defa kimlik bilgisi ve saniyede 2 mesaja kadar operasyon durumu ya da niyet raporlama mesajları gönderilmektedir [49].

4.1.2. ADS-B sinyallerinden veri seti oluşturulması

ADS-B sinyallerinden veri setleri oluşturulması bu çalışmanın en önemli bölümlerinden birisidir. Canlı yayınlanan sinyaller ile gerçek zamanlı olarak sistemin test edilmesi teknik olarak mümkün değildir. Sistemin doğru ve güvenilir çalıştığını kanıtlamak için gerçek zamanlı çalışan ikinci bir veri kaynağı gerekmektedir. Bunun için en geçerli kaynak Radar sistemleri olup teknik yetersizlikler nedeniyle herhangi bir Radar verisine erişilememiştir. İnternet üzerinden canlı olarak uçuş verilerini paylaştığını iddia eden bazı sitelere ulaşılmıştır (flightradar.com, flight24.com... vb.). Gerçekleştirilen teknik incelemelerde

dünyanın farklı coğrafyalarında gönüllü kişilerce bu çalışmada kullanılan benzer donanımlar kullanarak veri paylaşımında bulunduğu tespit edilmiştir. Söz konusu siteler tüm bu veriyi bir araya getirerek kendi web siteleri üzerinde WebSocket teknolojisi veya AJAX istekleri ile yayınlamaktadırlar. Bu yaklaşım nedeniyle herhangi bir hava taşıtının bilgilerinin yaklaşık 15 dakika gecikmeli olarak geldiği gözlemlenmiştir. Bu gecikme süresinin güvenlik nedeniyle kasten yaratıldığı da yine bu sitelerde belirtilmektedir. Gerçek zamanlı bir sistemin 15 dakika gecikmeli olarak gelen veriler ile test edilmesinin doğru bir yöntem olmayacağı kanaatine varılmıştır. Ayrıca bu kanaate varılmasına neden olan en büyük etken ise söz konusu yayın yapan sitelerin kendilerine veri gelmediği durumlarda ve özellikle uçaklar inişe geçtiği sırada gerçek veriyi değil kendi ürettikleri sentetik veriyi paylaştığı ve bu sürecin bir animasyona dönüştüğünün tespit edilmesidir. Bir hava taşıtının, bir hava limanından başka bir hava limanına seyahat, uçuş güzergâhları ve o bölgedeki hava koridorları gibi bilgiler internet üzerinden çok kolay bir şekilde elde edilebilmektedir. Bu bilgiler temin edildikten sonra çok basit bir yöntemle bir uçağın hangi zamanda nerede olacağı yaklaşık olarak hesaplanabilmekte ve veri gelmediği ya da hava taşıtı inişe geçtiği zamanlarda sentetik veri gönderimi yapılabilmektedir. Bu durum ve 15 dakika gecikmeli gelen verilerden ötürü gerçek zamanlı bir şekilde sistem testi ne yazık ki mümkün olmamıştır.

Sistemin test edilebilmesi için farklı bir çözüm geliştirilmiştir. Bu çözümde sinyaller bir ay süre ile kesintisiz olarak toplanmış ve veri toplama sistemi canlı olarak gözlemlenmiştir. 31 Ağustos 2015 ve 1 Ekim 2015 tarihleri arasında her bir anten için toplanan canlı veri hiçbir işleme tabi tutulmadan, en ham haliyle üç farklı dosyaya kayıt edilmiştir. Söz konusu verilerin yazıldığı her bir dosya yaklaşık olarak 66 GB'lık bir veri kümesi oluşturmaktadır. Toplanan verilerde zamanında gelmeyen ya da tekrarlı gelebilecek veriler olabileceği nedeniyle, test sonuçlarını etkilememesi için Apache Hadoop, Hive ve MapReduce tabanlı bir algoritma ile sinyal geliş zamanına göre sıralanmış ve kayıt sayısı bilgisi hesaplatılmıştır. Geciken ve tekrarlı gelen verilerin incelemesi aşamasında test sonuçlarının etkilenmesinin önüne geçilebilmiştir. Veri, ilerleyen bölümlerde açıklanacağı üzere, veri yolu üzerinde kontrollü bir şekilde geciktirilerek veya tekrarlı veri gönderilerek sistemin davranışı gözlemlenebilmiştir. Test yapılacak veri setinin elde edilmesi için son olarak yine MapReduce tabanlı bir algoritma ile 1-30 Eylül 2015 tarih aralığındaki veriler süzölmüştür. Bütün bu işlemler sonucunda farklı antenlerden gelen sinyaller için elde edilen dosyalar ve içerdikleri veri büyüklükleri Çizelge 4.2'de gösterilmiştir.

Çizelge 4.2. Toplanan verilerden oluşturulan veri setleri

Veri Seti	Veri Hacmi (GB)	Ham Kayıt Sayısı
Anten – 1	65,8	162 588 899
Anten – 2	65,8	162 588 896
Anten – 3	65,8	162 588 852

Veri setleri oluşturulduktan sonra test setlerinin(Ground Truth) oluşturulması gerekmektedir. Bu adımda Büyük Veri altyapısı kullanılarak sorgulanması istenecek analitik değerlerin hesaplanması ve dizinlenerek sorguya hazır hale getirilmesi sağlanmıştır. Analitik değerler hesaplanırken elde edilen veri 1 saniye, 1 dakika ve 10 dakikalık zaman pencereleri ile gruplanarak Hadoop üzerinde MapReduce ve Hive ile hesaplanmıştır. Sistemde kullanılmak için en uygun zaman penceresi olarak 1dk.lık pencere adımı seçilerek test seti oluşturulmuştur [51-52]. LM test edilirken sunum katmanından yapılacak sorgular ile bu test setlerindeki analitik değerler karşılaştırılarak Geri Getirme ve Kesinlik değerleri test/doğrulama alt yapısı tarafından otomatik olarak hesaplanmaktadır. Tasarlanan ve test edilecek LM'nin analiz sonucunda oluşturması istenen analitik metrikler 1 dakikalık zaman penceresi için aşağıdaki gibidir;

- Ortalama Tekil Uçak Sayısı
- Ortalama Tekil Bilinmeyen Uçak Sayısı
- Minimum Hava Taşıtı Hızı
- Maksimum Hava Taşıtı Hızı
- Minimum Hava Taşıtı Mesafesi
- Maksimum Hava Taşıtı Mesafesi
- Minimum İrtifa
- Maksimum İrtifa

Bu analitik metriklerin seçilmesinin özel nedenleri bulunmaktadır. Ortalama tekil uçak sayısı ve ortalama tekil bilinmeyen uçak sayısının hesaplanabilmesi gerçek zamanlı bir sistem için oldukça zor bir görevdir. Bunun nedeni AVK'nın bir durum kontrol mekanizmasına (state) ve arttırımsal bir algoritmaya sahip olacak şekilde geliştirilmesi gerekliliğidir. Belirli bir zaman penceresi aralığında hesaplanması gereken bu metrik, hafızada tutulması gereken özel bir veri yapısı gerektirmektedir. Bu tür durumlarda neredeyse tüm veri tabanları, performansı arttırmak için güvenilirlikten %2 civarında bir ödün vererek kardinalite kestirimi (cardinality estimation) algoritması olan HyperLogLog

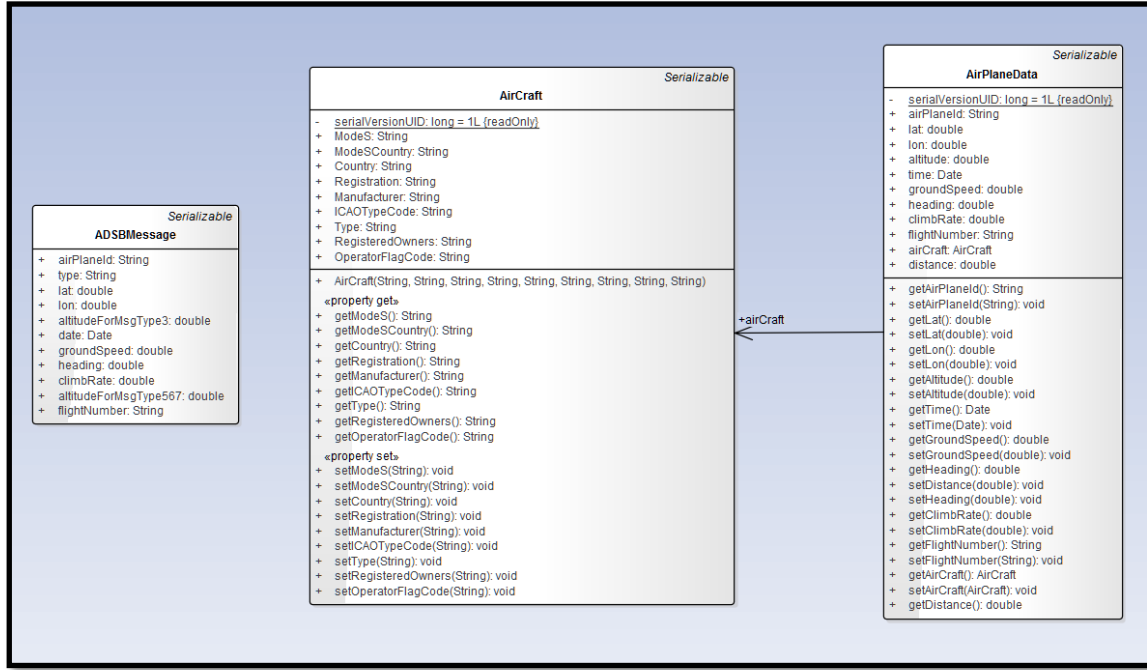
[53] algoritmasını kullanmaktadır. Bu durum çoğu durumda birçok uygulama için sorun yaratmasa da bu tasarım için sorun teşkil etmektedir. LM testini daha da zorlaştırmak, tam sonuç elde etmek ve sistem üzerinde baskıyı arttırmak için HyperLogLog algoritması hem sunum katmanında hem de test setleri oluşturulurken kullanılmamıştır. Gerçek ürün geliştirilmesi durumunda bu yaklaşım kesinlikle önerilmez. Bu çalışmada en kötünün daha kötüsü bir durumun oluşturulması için kullanılmıştır. Akışkan veri katmanlarında aşırı bellek ve işlemci kullanımına neden olsa da, test maksadıyla kullanılan donanım oldukça güçlü ve yüksek kapasiteli bir belleğe sahip olduğu için bir sorun yaratmamaktadır. Minimum ve Maksimum Hız/İrtifa analitik metrikleri, tek bir hava taşıtı bilgisinin doğru zamanda doğru şekilde işlenmemesi durumunda hatalı sonuç verebilmektedir. Bu nedenle Büyük Veri hacmine sahip çok hızlı akışkan verilerin işlenmesinde hata olup olmadığını ortaya koyacak bir analitik metrik olarak seçilmiştir. Minimum/Maksimum Mesafe metriği ise sistemin test edilebileceği son zorlama noktasıdır. Bu metriğin seçilmesinin nedeni mesafe hesabının gerçek zamanlı olarak yapılması ve bu hesaplamanın matematiksel denklemler ile yapılmasından ötürü işlemciye yük getirmesidir. Mesafe hesaplanması için Haversine formülü kullanılmıştır [51]. Söz konusu formül sonucunda dünya üzerinde iki coğrafi koordinat arasındaki (enlem, boylam) en kısa mesafe ve yön bilgisi elde edilir. Her bir hava taşıtı ve Bilkent Cyberpark'ta konuşlu anten ve yazılım tabanlı telsizler arasındaki Haversine mesafesi hesaplanmış ancak yön dikkate alınmamıştır. Bu hesaplama, veri işleme kapasitesi üzerinde baskı oluşturmaktadır. Söz konusu mesafe hesaplama formülü içerisinde sinüs, ark sinüs, karekök, derece – radyan dönüşümü ve bölme gibi matematiksel fonksiyonların hepsi kullanılmaktadır. Mesafelerin hesaplanmasından sonra, minimum ve maksimum mesafe 1 dakikalık pencere içerisinde tespit edilmektedir. Daha önce izah edildiği gibi bu hesaplama doğru zamanda ve doğru şekilde işlenmezse hataya neden olmaktadır. Bu analitik metrik LM'nin test edilmesinde en önemli metriktir. LM tarafından veri işleme sırasında en ufak bir hata yapılırsa, en kolay tespit edilebilecek analitik metriktir. Son ürün olarak kullanılan sistemler; genellikle kullanıcı sayısının gerçek zamanlı olarak sayılması, web sayfasına erişen tekil kullanıcı sayısının tespit edilmesi, bir makine öğrenimi modelinin sonucunun elde edilmesi gibi çok daha basit işlemlerde kullanılmaktadır. Bu durum göz önüne alındığında seçilen analitik metriklerin sistemin performansını ve güvenilirliğini gerçekten de test ediyor olduğu görülmektedir.

Yazılım tabanlı telsizlerden elde edilen verilere ait tekil uçak analitik metriklerinin hesaplanması için ayrıca bir işleme daha ihtiyaç duyulur. ADS-B sisteminden gelen verilerde her hava taşıtına ait bir ICAO kodu bulunmakta olup, bu kodun tekil olmadığı yani tek bir uçağı işaret etmediğı durumlarla karşılaşılabilir. Bu nedenle tekilliğı sağlamak için İnternette açık olarak bulunan ve herkesçe açık olarak paylaşılan bir SQLite taşınabilir veri tabanından faydalanılmıştır. Bu veri tabanı haftalık olarak güncellenmektedir. LM çalışmaya ilk başladığında, veri yolu aktif olmadan önce bu bilgi akışkan veri katmanı hafızasına yüklenmektedir. Gelen uçağın ICAO koduna göre tekil hafıza üzerinden sorgulanmasıyla hava taşıtının kimliklendirilmesi sağlanmıştır. Duran veri katmanı da benzer şekilde ilk çalışmaya başladığında yayın değışkeni (Broadcast variable) olarak Spark sürücü programı tarafından, Spark işçi (worker) düğümlere dağıtılmasıyla kullanılmıştır.

4.1.3. Veri modeli ve serileştirme

Sistemde verinin tek formatta hareket edebilmesi ve farklı teknolojilerin söz konusu veriyi işleyebilmesi için ortak bir veri modeli kullanılması uygun olacaktır. Aksi halde kullanılan her teknoloji için farklı veri işleme yöntemleri kullanılmış olacaktır. Bu sorunu ortadan kaldırmak için öncelikle gelen ADS-B mesajlarının tüm alanlarını ve tüm tiplerini içerecek şekilde genel bir ham mesaj modeli geliştirilmiştir. Yazılım tabanlı telsizlerden gelen ham veriler için “ADSBBMessage” sınıfından serileştirilebilir bir nesne kullanılmaktadır. Daha sonra, işleme esnasında, hava taşıtının kimliklendirilmesinin ardından “AirCraft” türünden bir nesne ile kimlik bilgileri saklanmaktadır. Verinin bu aşamadan sonra işlenmesi ve saklanması için ise “AirPlaneData” sınıfından serileştirilebilir bir nesne kullanılmaktadır. AirPlaneData nesnesi, yapısında bir alan olarak AirCraft nesnesini barındırmaktadır. Serileştirme için Apache Kryo yazılım çatısı (framework) kullanılmıştır. AVK ve DVK’da veri işleme amacıyla kullanılan Apache Spark teknolojisi de aynı serileştirme yazılım çatısını kullanmaktadır. Söz konusu yazılım çatısının en büyük özelliğı hızlı, etkin ve kolay kullanılabilir olmasıdır. Serileştirme ile bir Büyük Veri sisteminde kullanılan en temel işlem olan, çıkart dönüştür ve yükle (ETL) işlemleri oldukça hızlı ve etkin olarak gerçekleştirilebilmektedir. Ham haldeki veride, analiz için kullanılmayan ek bilgiler mevcuttur. Bunların ayıklanması ve sadece kullanılacak bilgilerin veri üzerinde kalması sistemin performansını doğrudan etkiler. Bu nedenle ikinci bir sınıfa ihtiyaç duyulmuştur. Ayrıca ham veriden gereksiz alanlar süzülerek işlenecek temel veri oluşturulurken, hava

taşıtı daha önceki bölümde belirtildiği gibi teşhis edilerek temel veri ek bilgileri zenginleştirilmektedir. Şekil 4.2’de temel ETL işlemi için kullanılan sınıf diyagramları gösterilmektedir.



Şekil 4.2. ADS-B Mesaj formatları için sınıf diyagramları

Resim 4.4’de ham verinin işlenerek ve zenginleştirilerek ETL işlemine tabi tutulmasından sonra elde edilen ve LM’de analiz için kullanılan verilerin çok ufak bir kısmı gösterilmektedir. Apache Kryo ve Jackson yazılım çatıları vasıtasıyla söz konusu veriler JSON formatına dönüştürülerek görselleştirilmiştir. Zenginleştirme işleminden sonra Resim 4.4’de de görüldüğü gibi hava taşıtının ICAO kodu kullanılarak, ait olduğu ülke, uçağın modeli, kullanıldığı hava yolu şirketi bilgileri, o anki uçuş koordinatları, yönü, irtifası, uçuş numarası, operatör kodu bilgileri elde edilmiştir. ICAO kodu kullanılarak hava taşıtının tanımlanamadığı durumlarda bu bilgiler “UNK” olarak atanmakta ve analizlerde bilinmeyen hava taşıtı olarak incelenmektedir.

Resim 4.4. ETL işleminden sonra analitik olarak işlenecek veriler

4.2. Lamda Mimari Bileşenleri

Lamda Mimarının oluşturulabilmesi için temel olarak duran veri katmanı, akışkan veri katmanı ve sunum katmanının kullanılması yeterli olmaktadır. Ancak LM'nin çalışması esnasında; AVK ve DVK'nın koordineli bir şekilde çalışması, sistemin izlenmesi ve kendine ait bir veri yolunun bulunması gerekmektedir. Lamda Mimarının sadece üç katmandan oluştuğu belirtilse de, bu ek yapı alt bileşenleri de bünyesinde barındırmaktadır. Lamda Mimarının birçok bileşeni arasında etkileşim bulunmaktadır. Bu bileşenlerin her biri detaylı olarak ilerleyen bölümlerde detaylı olarak anlatılmıştır.

4.2.1. Veri yolu (Data Bus) katmanı

Veri yolu katmanının, gerçek durum kullanımında RadarCape, Odroid X4 single board bilgisayar ve USB RTL-STR yazılım tabanlı telsizler kullanılmıştır. RadarCape ADS-B sinyallerinin gerçek zamanlı olarak çözülmesi ve farklı servisler ile gönderilmesini sağlayan Linux işletim sistemi tabanlı bir donanımdır. İki adet Odroid X4 ve RTLSDR, USB ara birimine sahip yazılım tabanlı telsizde kullanılarak sinyaller çözülmemektedir. Her üç donanımda da aktif diyapazon antenler kullanılmıştır. Tüm ADS-B sinyal çözümleyiciler ile TCP 3000 portu kullanılarak veri transferi gerçekleştirilebilmektedir. Kafka-TCP Soket yazılımı ile bu porta erişim sağlanarak veri Kafka kümesine (cluster)

geçirilmektedir. Kafka Kümesi sadece kendisine ait donanımlar üzerinden veri transferi gerçekleştirmektedir. Veri üretici yazılımı, Kafka Broker ile 10 bölüntü (partition) kullanacak şekilde oluşturulan “ADSBGercek” ve “ADSBTest” başlıkları (kafka topic) üzerinden veriyi AVK ve DVK’ya göndermektedir. 20 bölüntü (partition) sayesinde yüksek hızlarda gelen akışkan verinin Kafka üzerinden paralel (20 kanal) olarak gönderilmesi sağlanmaktadır. Sistem uygun donanım sağlanması ile, Kafka Broker üzerinden geçen her verinin 1 yıl boyunca silinmemesini ve bu süre içerisinde gelecek yeniden gönderim (replay) isteklerine cevap verilmesini sağlayacak şekilde ayarlanmıştır. Kafka bölüntüler ve Broker replikasyon olmadan çalışacak şekilde ayarlanmıştır. Veri üretici yazılımı, ayrıca Log4J2 yazılım çatısı üzerinden gerçekleşen olayları sistem izleme katmanına göndermektedir. LM çalışmaya başladığında, YKK’dan gelen başlangıç parametrelerine göre ADSBGercek ya da ADSBTest başlıklarına veri gönderilmeye başlanmaktadır. AVK, yine bu Kafka başlıklarından birisine kayıt olarak, veriyi paralel şekilde üzerine alıp işlemektedir.

DVK için “Camus” teknolojisi kullanılarak, gelen veri belirli aralıklarda HDFS’e yazılır. Camus, LinkedIn tarafından Gobblin olarak yeniden geliştirilmiştir ancak henüz olgunlaşmış bir teknoloji olmamasından dolayı bu çalışmada yine de Camus kullanılmıştır. Camus basitçe bir MapReduce iş parçacığı olduğu için, her çalıştırılmasında, en son kaldığı zaman aralığından itibaren veriyi zaman damgasına göre ayırarak HDFS’e farklı bölümlere yazar. Camus üzerinden veriyi HDFS’de yazabilmek için gelen verinin yapısı ve zaman damgasının çözülmesi gerekmektedir. ADS-B mesajları farklı tipte olduğundan ötürü genel bir veri modeli üretilmiş ve JSON formatında kullanılmıştır. Bunun için com.linkedin.camus.coders.MessageDecoder sınıfından türetilen bir çözücü geliştirilmiştir. Bu çözücü ile Kafka’dan gelen verinin DVK’da işlenirken performans kaybına neden olmaması için HDFS’e yazılmadan önce ADS-B mesaj yapısının çözülmesi, gereksiz (işlenmeyecek) ADS-B mesaj tiplerinin süzülmesi ve kullanılacak bilgi alanlarına göre bir JSON nesnesine dönüştürülerek yazılması sağlanmıştır. Veri işlenirken ilk ETL işlemi bu noktada uygulanmıştır. YKK tarafından Camus MapReduce iş parçacığı (MapReduce Job) dakikada bir defa çalıştırılmakta ve veriler HDFS’e yazılmaktadır. Camus için gerekli tüm çalışma zamanı parametreleri yapılandırma dosyaları aracılığı ile gerçekleştirilmiştir.

4.2.2. Duran veri katmanı

DVK temel olarak, canlı veri işleme durumunda oluşabilecek hataların düzeltilmesi amacıyla LM’de kullanılmaktadır. Duran veri katmanı, Kafka Broker üzerinden canlı olarak gelen verilerin “Camus” entegrasyon yazılımı kullanılarak Hadoop dağıtık dosya sistemi (HDFS)’e yazılması ile duran hale gelen veriler üzerinde çalışmaktadır. YKK katmanı tarafından belirli zamanlarda DVK’nın çalıştırılmasıyla yeni gelen veriler işlenerek SK’da çalışmakta olan Druid’e iletilmektedir.

DVK katmanında, Hadoop YARN üzerinde, kümeleme modunda çalıştırılan Apache Spark teknolojisi etkin biçimde kullanılmaktadır. HDFS’e yazılan veriler belirli aralıklarla okunarak hafızaya alınmakta ve RDD olarak artırımlı (incremental) biçimde hafıza üzerinde biriktirilmektedir. RDD’ler öncelikle hafızada, yeterli yer kalmadığında diskte saklanacak şekilde (MEMORY_AND_DISK_2) ayarlanmıştır. Verilerin etkin biçimde işlenebilmesi için RDD’ler üzerinde yeniden dengeleme ve sıralama işlemi (repartitionAndSortWithinPartitions) operasyonu YKK tarafından yapılan ölçümler sonucunda gerçekleştirilmektedir. Bu operasyon yapılmadığı takdirde sistemin işleyişi üzerinde ciddi performans kayıpları meydana gelmektedir.

Verilerin DVK’da işlenmesi için DataFrame ve SparkSQL modülleri kullanılmıştır. SparkSQL dağıtık sorgulama motoru olarak DataFrame veri yapıları ile çalışmaktadır. Detaylı kod geliştirmek yerine istenilen hesaplamaların kolayca SQL sorguları olarak yazılmasını sağlamaktadır. DVK’da kullanılacak RDD’ler oluşturulduktan sonra verilerin işlenmesi için SQLContext nesnesi yaratılmıştır. DataFrame üzerinden verilerin işlenebilmesi için ADSBMessage nesnesi türünde bir sınıf oluşturularak şema olarak tanıtılmıştır. SQL sorguları söz konusu DataFrame’ler üzerinde çalıştırılarak istenilen analitik hesaplar tamamlanmakta ve JSON dosyası olarak yine HDFS’e yazılmaktadır. HDFS’e yazma işlemi biter bitmez YKK tarafından sonuç JSON dosyasının boyutu kontrol edilmekte ve eğer boyut 1 GB’dan küçükse Druid REST API komutuyla Druid OverLord servisine, değilse YKK tarafından çalıştırılan HadoopDruidIndexer MapReduce iş parçacığı ile SK’ya iletilmektedir. Druid yeni gelen veriyi kendi yapılandırma dosyasında belirtilen şema ve ayarlara göre (spec_file) ithal ederek dizinleme yapmakta (indexing) ve kendi veri saklama yapısı olan kesitleri (segments) oluşturmaktadır. Oluşturulan kesitler Druid History düğümlere gönderilerek sorguya hazır hale getirilmektedir. Bu süreçte çeşitli

nedenlerle oluşan hatalar YKK tarafından tespit edilerek, veri ithali ve DVK veri işleme işlemleri yeniden çalıştırılmaktadır. Arka arkaya üç defa hata alınırsa sistemin çalıştırılması tamamen durdurulmaktadır.

DVK'nın çalışması ve verinin SK'ya iletilmesi başarılı şekilde tamamlandıktan sonra, SK'da AVK'nın göndermiş olduğu ve DVK tarafından aynı zaman aralığında işlenen veriler boşaltılarak AVK üzerindeki veri yükü azaltılmaktadır. DVK'nın çalışması tamamlandıktan sonra sisteme geciken ya da tekrarlı olarak gelen veriler nedeniyle oluşan hatalar tamamen düzeltilmektedir. Druid koordinatör (coordinator) REST uygulama programlama arayüzü (API) üzerinden bu işlem takip edilmektedir.

Bu çalışmanın en özgün ve farklı yanlarından birisi olan AVK ve DVK'nın farklı teknolojiler yerine aynı teknolojiyi kullanılarak geliştirilmiş olmasıdır. Lamda Mimari üzerinde aynı hesaplamaları ve işlemleri yapacak yazılım kodunun AVK ve DVK için farklı teknolojilerde geliştirilmesi yerine, aynı teknolojinin kullanılmasıyla bir defa kodlanması, iki farklı teknoloji için kod geliştirme, test etme ve bakım işlemlerinin ortadan kaldırılmasına olanak tanımaktadır. Büyük Veri işleme uygulaması için, Apache Spark teknolojisinin kullanılmasıyla geliştirilen yazılım kodları, her iki katmanda da birebir aynı biçimde kullanılmıştır.

Hava taşıtlarının kimliklerinin belirlenmesi için internette kolayca bulunabilen bir SQLite veri tabanı kullanılmıştır. Söz konusu veri tabanı içeriğinde hava taşıtlarının ICAO kodlarına göre taşıta ait operatör, ülke, model gibi gerek duyulan tüm bilgiler mevcuttur. Bu bilgiler kullanılarak gerçek zamanlı ve toplu veri işlemlerde hava taşıtı kimliklendirme işlemleri gerçekleştirilebilmektedir. Ancak çalışma anında bu veri tabanının sürekli olarak sorgulanması için tüm Spark kümelemesinde (cluster) kullanılabilir olması gerekmektedir. Bu veri tabanının merkezi çalışacak şekilde tüm kümelemenin erişebileceği bir yerde kurulup bu biçimde kullanımı mümkün olsa da, çalışma anında sürekli olarak sorgulanması veri tabanına aşırı yük getirecektir. Ayrıca merkezi bir yerden sürekli olarak sorgulanması performans kayıplarına yol açabilmektedir. Bu nedenle söz konusu veri tabanı metin dosyasına dönüştürülerek çalışma anında Spark teknolojisinde bulunana yayınlama değişkeni (broadcast variable) olarak tüm kümelemeye dağıtılmaktadır. Kümeleme üzerinde çalışan yazılım ilk defa çalışmaya başladığında tüm veriyi hafızasına alarak bir anahtar-değer sözlüğü (key-value dictionary) olarak saklamakta ve doğrudan hafıza

üzerinden sorgulamaktadır. Bu şekilde performans kayıplarının ve söz konusu veri tabanının aşırı sorgulama yükü altında çalışmama durumundan dolayı sorun yaşanmasının önüne geçilmiştir.

4.2.3. Akışkan veri katmanı

Akışkan veri katmanı, Kafka Broker'dan gelen verilerin üzerinde ETL ve analitik işlemler yaptıktan sonra sonuçları SK'ya ve aynı zamanda gerçek zamanlı uygulamaya göndermektedir. Bu katmanda Spark Streaming teknolojisi kullanılmıştır. Spark Streaming aynı anda veri yolundan gelen verileri 1 saniyelik mikro-toplu (Micro Batches) işleme yöntemi ile sunum katmanına ve gerçek zamanlı uygulamaya göndermektedir. Bu nedenle hem veri yolu hem de SK arasında entegrasyon işlemleri gerçekleştirmek için geliştirme ve yapılandırma yazılımları kodlanmıştır. SK'da kullanılan Druid veri tabanının doğrudan (native) Spark Streaming desteği yoktur. Bu nedenle Druid Tranquility Direct API kullanılarak OverLord veri ithal servisi için özel bir yazılım geliştirilmiştir. Druid kendi yapılandırma dosyasındaki değerlere göre veri ithal işlemini gelen veri için otomatik olarak Tranquility servisi ile gerçekleştirmektedir.

Verinin Spark Streaming teknolojisi ile güvenli bir şekilde alınabilmesi için spark-kafka yazılım çatısı kullanılmıştır. Kafka üzerinden verinin hızlı bir şekilde alınabilmesi için Kafka Producer'ın veriyi 20 bölüntü ile göndermesi ayarlanmış ve bu nedenle Spark Streaming alıcısının da buna uygun olarak yapılandırılması sağlanmıştır. Kafka'dan Spark Streaming Receiver'a veriler DStream olarak belirlenen bir zaman penceresinde gönderilmektedir. Sistem çapındaki gecikmenin minimumda tutulması için zaman penceresi ve zaman aralığı 1 saniye değerine ayarlanmıştır. Bu şekilde; geleneksel uygulamalarda yer alan kayar pencere yaklaşımı kullanılmayacaktır. Her saniye için gelen DStream RDD'ler toplu olarak işlenerek, SK ve gerçek zamanlı uygulamaya iletilmektedir. AVK'da verinin alınması ve işlenmesi için KafkaInputStream nesnesi kullanılmıştır. Kafka'dan verilerin okunabilmesi için batchInterval ve spark.streaming.blockInterval parametreleri sırasıyla 1s, 50 ms olarak ayarlanmıştır. Bu yüksek hızlı ve hacimli veriyi en etkin biçimde okumak için okunması gereken başlık (topic) ve her birisinde 1 adet iş parçacığı bulunan (thread) 20 adet DStream tüketici (consumer) yapılandırılmıştır. Verinin veri yolundan paralel olarak okunması ve okuma işleminden farklı bir paralellikte işlenmesi gerekmektedir. Başka bir deyişle verinin okunması ile işlenmesi farklı sayıda

paralel işlem parçacıkları ile yapılmaktadır. Verinin işlenmesinde daha fazla sayıda paralelleştirmeye ihtiyaç duyulduğu için gelen DStream'ler sıralama işleminden sonra birleştirme(union) yapılarak bir UnionDStream oluşturulmuş ve bu aşamada RDD bölüt sayısı 40 olarak belirlenmiştir. Bu şekilde, veri üzerinde Spark ile 40 paralel iş parçacığı kullanılarak işlem yapılmaktadır. Veri kaybının önüne geçilebilmesi için “Write Ahead Logs” özelliği aktif hale getirilmiştir. Her verinin tam olarak bir defa okunabilmesi için güvenilir alıcı yapısı kullanılmıştır (Reliable Receiver).

Veri yolundan elde edilen verilerde ADS-B mesajları bulunmaktadır. Daha önce de belirtildiği gibi ADS-B mesajları farklı amaçlar için farklı formatlarda yayınlanmaktadır. Bu nedenle analizlerde mesajların tümü kullanılmayacaktır. Genel olarak, hava taşıtının göndermiş olduğu tekil kimlik belirteci, hız, irtifa, yön ve sefer numarası dışındaki alanları içeren mesajlar AVK'da süzülerek üzerinde işlem yapılmamaktadır. Gelen veriler üzerinde yapılan süzme işleminden sonra, verinin hava taşıtı tarafından yayınlanma zamanına ve hava taşıtının koordinat ve kimlik belirtici alanlarına göre indirgeme yapılarak çoklu mesajlar tek bir mesaja dönüştürülmektedir. Ayrıca hava taşıtı koordinatları ile referans merkezi arasındaki mesafenin Haversine formülü ile hesaplanması işlemleri de gerçekleştirilmektedir. Tüm hesaplamalar tamamlandıktan sonra veri doğrudan SK'ya iletilmektedir.

Analiz, indirgeme ve kimliklendirme işlemleri tamamlanan veriler, Spark Streaming üzerinden gerçek zamanlı uygulama veri tabanı olan Redis'e ayrıca kayıt edilmektedir. Redis bu anlamda yayınlı-kayıtlı ol yöntemi ile çalıştığından ötürü entegrasyon için kendisine ait uygulama geliştirme yazılım çatısı kullanılmıştır. DStreamRDD içerisindeki her bir veri Redis API ile veri tabanına yazılmaktadır. Ancak bu durumda Redis'le kurulması gereken bağlantının RDD bölütleri içerisinde gerçekleştirilmesi gerekmektedir. Aksi takdirde Redis bağlantısının serileştirilerek her bir RDD bölütüne iletilmesi gerekmektedir. Bu teorik olarak mümkün olsa da pratikte etkin bir çözüm değildir. Bu nedenle bir bağlantı kontrol mekanizması ve singleton nesne üzerinden bu işlem gerçekleştirilmektedir. Her bir RDD bölütü içerisinde bağlantı yönetimi yapılarak veriler etkin biçimde Redis veri tabanına yazılmaktadır. AVK ile veri yolu arasında Kafka'dan veri okunması işlemi Apache ZooKeeper teknolojisinin koordinesi ile gerçekleştirilmektedir.

Bu çalışmanın en özgün yanlarından birisi olan AVK ve DVK'nın farklı teknolojiler ile kodlanması yerine aynı teknolojinin kullanılıyor olmasıdır. Aynı hesaplamaları ve işlemleri yapacak yazılım kodunun AVK ve DVK için farklı teknolojilerde kodlanması yerine bir defa kodlanarak kullanılması, iki farklı teknoloji için kod geliştirme, test etme ve bakım işlemlerinin ortadan kalmasını sağlamaktadır. İki katmanda da aynı mantığın kullanılıyor olması nedeniyle, Apache Spark teknolojisi üzerinde geliştirilen yazılım kodları, bir defa kodlanarak her iki katmanda da kullanılmıştır.

Hava taşıtlarının kimliklerinin belirlenmesi için internette kolayca bulunabilen bir SQLite veri tabanından faydalanılmıştır. Söz konusu veri tabanı içeriğinde hava taşıtlarının ICAO kodlarına göre operatör, ülke, model gibi gerek duyulan tüm bilgiler mevcuttur. Bu bilgiler kullanarak gerçek zamanlı ve toplu veri işlemlerde hava taşıtı kimliklendirme işlemleri gerçekleştirilebilmektedir. Ancak bu veri tabanının çalışma anında sürekli olarak sorgulanması için tüm Spark kümelemesinde (cluster) kullanılabilir olması gerekmektedir. Bu veri tabanının merkezi bir çalışacak şekilde tüm kümelemenin erişebileceği bir yerde kurulup bu şekilde kullanımı mümkün olsa da, çalışma anında sürekli olarak sorgulanması veri tabanına aşırı yük getirecektir. Ayrıca merkezi bir yerden sürekli olarak sorgulanması performans kayıplarına yol açabilmektedir. Bu nedenle söz konusu veri tabanı metin dosyasına dönüştürülerek çalışma anında Spark Streaming aktif olduğu anda dosyadan yüklenmekte ve hafızada bir sözlük (dictionary) olarak saklanmaktadır. Bu şekilde performans kayıplarının ve aşırı sorgulama yükü altında söz konusu veri tabanının çalışmama durumundan dolayı sorun yaşanmasının önüne geçilmiştir.

4.2.4. Sunum katmanı

Sunum katmanı tamamen kendi başına çalışan bir kümeleme alt yapısı üzerinde çalıştırılmaktadır. Veri tabanı servisleri bu alt yapı üzerinde farklı sunucular üzerinde çalışacak şekilde kurulmuştur. SK aynı anda iki farklı katmandan veri almaktadır. Her bir katmandan gelen verilerin özellikleri birbirlerinden farklı olduğu için veri tabanına farklı şekilde ithal edilmektedir. Verilerin ithal edilmesi için JSON formatında yapılandırma dosyası olarak hazırlanan veri şemaları kullanılmaktadır. Söz konusu şema DVK ve AVK için oldukça benzer özellikler göstermekte olup farklılık gösteren kısmı ise veri geliş kaynaklarının katmanlara göre farklı olarak belirtilmesidir. Veri tabanına ithal için temelde üç adet veri şeması yapılandırma dosyası tanımlanmıştır. Bunlardan iki adedi DVK için bir

adedi de AVK için oluşturulmuştur. DVK da iki adet veri şemasının tanımlanmasının nedeni 1 GB'ı geçen dosya boyutları için verilerin DruidHadoopIndexer MapReduce iş parçası, küçük olan dosya boyutları için ise OverLord veri dizinleme servisinin kullanılıyor olmasıdır. AVK da ise sadece OverLord veri dizinleme sistemi kullanılmıştır.

Veri şemasında kullanılan boyut değişkenleri (dimensions), “ModeS” ve “flightNumber” alanları olarak yapılandırılmıştır. Metrik değişkenleri ise “altitude”, “groundSpeed”, “Distance” alanları olarak seçilmiştir. Gelen hava taşıtı kimliklerinin tekillik sağlayabilmesi için “ModeS” ve “flightNumber” alanları birlikte boyut değişkeni olarak kullanılmıştır. Bu veri şemasında veriler üzerine uygulanan bir araya getirme (aggregator) işlemleri Çizelge 4.3’de gösterilmektedir. “ModeS” ve “flightNumber” metrikleri için kardinalite (cardinality) kestirimi ve ortalama işlemi, “groundSpeed” metriği için Min ve Max işlemleri, “altitude” metriği için Min ve Max işlemleri, “Distance” alanı için Min ve Max işlemleri uygulanmıştır. Zaman çözünürlüğü 1 dakika, bölüt çözünürlüğü ise 10 dakika olarak ayarlanmıştır. Bu şekilde tüm veriler bölütlerde birer dakikalık bir araya getirme işlemi ile saklanmaktadır. Bölütlerin oluşturularak derin saklama alanlarına gönderilmesi on dakika da bir gerçekleşmektedir. Ayarlanan zaman penceresi çözünürlüğünde (bir dakika) her bir metrik için istenen işlemler veri tabanına ithal edilirken gerçekleştirilmektedir. Veriler veri tabanına yazılmak üzere geldiği andan itibaren sorgulanabilir olmaktadır.

Çizelge 4.3. Veri boyut değişkenleri ve uygulanan işlemler

Veri Boyut Değişkeni	İşlem (Bir Araya Getirme)
ModeS ve Fight Number	Cardinality Estimation ve Mean
Ground Speed	Min ve Max
Altitude	Min ve Max
Distance	Min ve Max

Çizelge 4.3’de belirtilen veri boyut değişkenleri ve uygulanan işlemler, DVK veri ithal şeması için de aynıdır. Aynı veri şemasında, verilerin veri modelindeki “time” alanının zaman dizinlemesi için kullanılacağı da belirtilmektedir. Zaman damgası olarak “time” alanı kullanılacağı veri tabanına verilerin ithal edilmesinde çok büyük bir öneme sahip olmakta ve tüm işlemler bu zaman damgası alanı kullanılarak gerçekleştirilmektedir.

AVK'da gelen veriler için gerçek zamanlı ithal yapılmaktadır. Spark Streaming teknolojisinin Druid desteği yoktur. Bunun için Druid Tranquility Direct API uygulama programlama arayüzü [44] kullanılarak bir entegrasyon yazılımı geliştirilmiştir. Bu yazılımda veri kaynağının AVK olduğu belirtilerek, boyut değişkenleri ve metrikleri veri şemasındaki alanlar ile uyumlu olacak şekilde tanımlanmıştır. Ayrıca veri ithalinin kümelemede bulunan ZooKeeper kullanılarak koordine edileceği belirtilmiştir. Bu şekilde gelen veriler, gerçek zamanlı olarak, yüksek hızlarda ve istenen zaman çözünürlüğünde bir araya getirme işlemleri (aggregation) uygulanarak veri tabanına yazılmaktadır.

DVK için daha önce de belirtildiği gibi iki farklı veri şeması ayar dosyası hazırlanarak kullanılmıştır. 1 GB'tan büyük duran veri dosyaları DruidHadoopIndexer iş parçacığı [45] ile Hadoop üzerinde işlenerek veri tabanı için bölütlerin oluşturulması sağlanmaktadır. Eğer işlenecek dosya 1 GB'tan düşük boyutlu ise doğrudan Druid OverLord servisi ile veri ithal işlemi gerçekleştirilmektedir. Bu katmanın veri işleme ve ithal etme mekanizmasının esas amacı AVK'da oluşan sorunları geriden gelerek (daha sonraki bir zamanda) düzeltmek olduğu için belirli zamanlarda çalıştırılması gerekmektedir. Bu işlem YKK tarafından, AVK üzerindeki yükün ve elde edilen bazı ölçümlerin kullanılması yardımıyla, DVK'nın çalıştırılma kararının verilmesi sonucunda gerçekleştirilmektedir. SK, YKK tarafından veri tabanı kümelemesi üzerinde çalışan tüm servisler ile REST API kullanılarak haberleşme ve üretilen günlüklerin takibi yönetilmektedir.

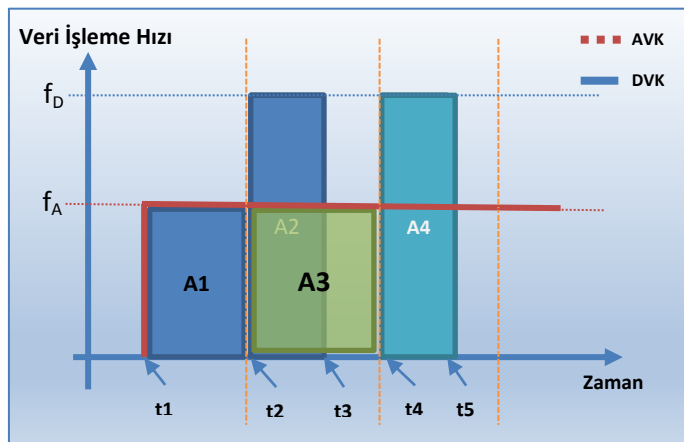
SK'nın Lamda Mimari çalışma prensiplerine uygun olarak işletilebilmesi için DVK veri işlemlerini tamamladığında, AVK'dan DVK'nın işlediği veri miktarı kadar verinin silinmesi gerekmektedir. Bu nedenle, veri tabanı kodları bu işlemin çok hızlı ve etkin biçimde yapabilmesi için, ek özellikler kazandırılması maksadıyla büyük ölçüde değiştirilip yeniden derlenerek kullanılmıştır. Açık kaynaklı teknoloji kullanılarak yeniden oluşturulan SK veri tabanı, orijinal veri tabanı ile benzer özellikler taşısa da, kazandırılan bu yeni özellikler ile performansı oldukça artırılmış ve etkin olarak LM'de kullanılmıştır.

Bu çalışmada Lamda Mimarinin geciken verileri ne kadar zaman içerisinde ve ne kadar başarı oranı ile düzelttebildiğinin net olarak tespit edilebilmesi için, geciken verilerin hiçbir şekilde işleme alınmaması, yapılandırma dosyasında belirtilerek ayarlanmıştır (Rejection Policy). Geciken veri işleme politikası tasarım zamanında verilmesi gereken bir karardır. Ancak Lamda Mimarinin en önemli amaçlarından birisi, akışkan veri katmanındaki

sorunlardan kaynaklanan hataların düzeltilmesidir. Akışkan veri katmanına gelen verilerin aynısı duran veri katmanında da bulunmaktadır. Bu verinin akışkan veri katmanına göre daha uzun sürede ve daha farklı zamanlarda işlenmesi sonucu hatalar giderilmektedir. Ancak söz konusu verinin gerçek zamanlı ithali durumunda veri reddetme yaklaşımı benimsenerek işlenmesi ya da geç gelen verinin işlenmesi yaklaşımı tasarım zamanında netleştirilmesi gereken bir karardır. Bu çalışmada geç gelen verilerin akışkan veri katmanı tarafından reddedilmesi, verinin işlenmesinin daha sonraki bir zamanda duran veri katmanı tarafından gerçekleştirilerek eksik verinin tamamlanması yaklaşımı benimsenmiştir.

4.2.5. Yönetim/koordinasyon katmanı

Yönetim/Koordinasyon katmanı Lamda Mimarinin en önemli katmanlarından birisidir. Bu katmanın görevi, Temelde sistemde gerçekleşen olayları, AVK ve DVK'nın veri işleme esnasında hafıza ve CPU yüklerini takip ederek gerekli zamanlarda DVK'nın çalışmasını sağlamak, veri ithal işlemlerinin sonuçlarını kontrol etmektir. Ayrıca Lamda Mimarinin en önemli özelliklerinden birisi, değişmeyen (immutable) veri setleri üzerinde yeni analizler yapılması ve eski analiz sonuçlarında ya da kullanılan algoritmaların değişmesi nedeniyle yeni analiz gerekmesi durumunda DVG ve AVG'lerin oluşturulmasını da yönetmesidir. YKK, DVK'yı gerektiği zamanlarda başlatabilmek için AVK, DVK ve veri yolu alt sistemlerini sürekli olarak takip etmektedir. Sistemin temel tasarımı veri yolundan gelen veri akış hızına göre gerçekleştirilmiştir. Şekil 4.3'de teorik olarak çalışması modellenen AVK ve DVK'lara ait veri işleme hızı – zaman grafiği gösterilmiştir.

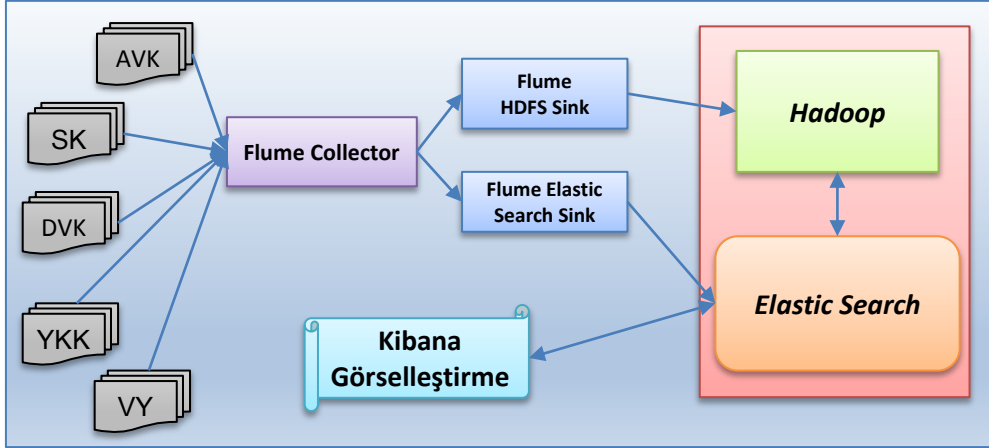


Şekil 4.3. Veri Akış Hızı - Zaman değişimi

Burada f_D duran veri katmanı veri işleme hızını, f_A akışkan veri katmanı veri işleme hızını, t_1 - t_5 ise sistemin çalışmasında önemli zamanları temsil etmektedir. (t_1, t_2) zaman aralığı sadece AVK'nın çalıştığı, (t_2, t_3) AVK ve DVK'nın aynı anda çalıştığı ilk zaman aralığını, (t_3, t_4) sadece AVK'nın çalıştığı zaman aralığını ve (t_4, t_5) yine AVK ve DVK'nın birlikte çalıştığı diğer zaman aralığını göstermektedir. Burada gösterilen f_A ve f_D sistemin çalışma yeteneğine göre deneysel olarak belirlenebilen parametrelerdir. Tasarlanan sistem için yapılan denemelerde f_A 'nın maksimum değerinin ortalama 56000 veri/s, f_D 'nin ise 130 000 veri/s olduğu tespit edilmiştir. Normal çalışma durumunda ise ortalama f_A 25000 veri/s, f_D ise 54000 veri/s olarak ayarlanmıştır. Bu değerlere ek olarak (t_2, t_3) ve (t_4, t_5) zaman aralıklarının 342 s olması ve (t_1, t_2) ve (t_2, t_4) zaman aralığının 700 s olması YKK üzerinde ayarlanmıştır. Bu parametrelerden de görüldüğü gibi A_1 , A_2 , A_3 ve A_4 hesaplandığında $A_1 < A_2$ ve $A_3 < A_4$ koşulunun sağlandığı, Böylece sistemin güvenli şekilde veri işlemesi ve veri işleme süresinin azami 6 dakika olaması belirlenmiştir. AVK'da yaklaşık 12000000 veri biriktiğinde DVK'nın çalıştırılmaya başlanacağı öngörülmektedir. Bu durumun matematiksel olarak modellenmesi çok doğru sonuçlar vermemektedir. Kümelemenin (cluster) çalışma ve kaynak tüketimi doğrusal ya da doğrusal olmayan şekilde modellenememektedir. Devasa bir kümeleme içerisinde binlerce iş parçası (thread) çalışırken birbirleri ile etkileşmesi, sunucuların kaynak durumlarının takibinde oluşabilecek sıkıntılar ve veri akış hızlarının her zaman aynı olmaması gibi nedenler bu modelin geliştirilmesini oldukça zorlaştırmaktadır [45].

4.2.6. Sistem izleme alt yapısı

Şekil 4.4'de Lamda Mimari için sistem izleme alt yapısı gösterilmiştir. Lamda Mimari içerisinde bulunan her bir katman ve yazılım Log4J2 yazılım çatısını kullanarak gerekli metrikleri Apache Flume Collector servisine iletmektedir.



Şekil 4.4. Sistem izleme alt yapısı

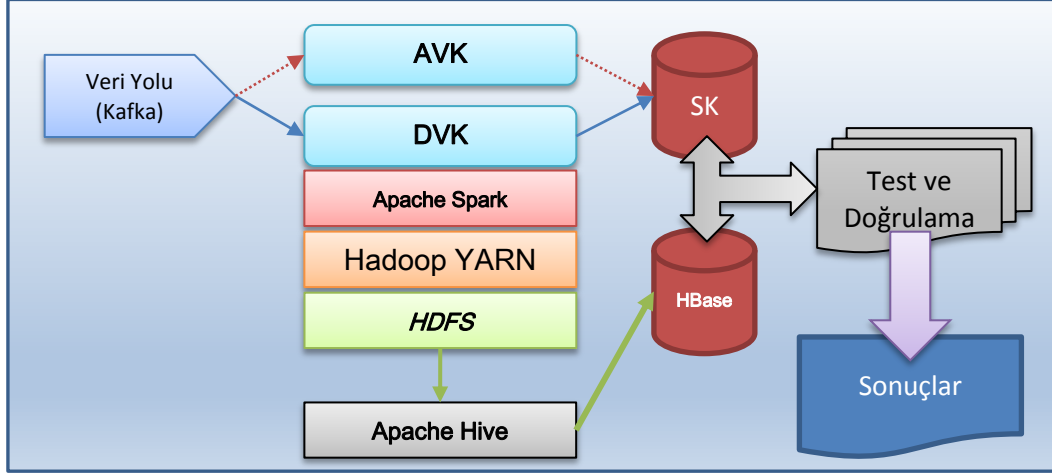
Apache Flume Collector servisinden veri iletilirken Interceptor servisinde olay kayıtları ve metrikler üzerinde ETL işlemleri uygulanmaktadır. Daha sonra tüm bu veriler bir taraftan Hadoop HDFS'ye yazılırken bir taraftan da Elastic Search'e dizinlenmek üzere iletilmektedir. Elastic Search teknolojisi gelen verileri dizinleyerek biriktirmektedir. Bu veriler üzerinde Kibana ile görselleştirme işlemleri uygulanmakta ve sistemin canlı olarak takip edilmesi sağlanmaktadır. AVK, DVK ve VY olay günlüklerinin yanında işledikleri veri miktarı, kullandıkları sistem kaynağı gibi bilgileri de ayrıca göndermektedir. Bu bilgiler bir taraftan da YKK'ya iletilmekte ve DVK'nın başlatılması için gerekli kararın verilmesine destek olmaktadır.

Sistem izleme alt yapısı özellikle, test/doğrulama sistemi ile Lamda Mimari test edilirken oluşabilecek hata ve sorunları tespit etmek için kullanılmaktadır. Sistemde oluşan hatalar giderilerek yeniden test işlemine başlanmaktadır. Testler sırasında oluşan hataların düzeltilerek tekrarlanması ile aynı koşullarda elde edilen sonuçların karşılaştırılması mümkün olmaktadır.

4.2.7. Test/doğrulama altyapısı (Ground Truth)

Daha önce belirtildiği gibi, test/doğrulama alt yapısı Hadoop üzerinde MapReduce yöntemi ile hesaplanmış veriler kullanılarak, LM'deki SK'nın sorgulandığı bir yazılımdır. SK'daki verilerin doğrulanabilmesi için Apache Hive teknoloji ile SQL tabanlı sorgular oluşturulmuştur. Bu sorgular otomatik olarak MapReduce iş parçalarına dönüştürülerek sonuçlar oluşturulmaktadır. Temel olarak AVK ve DVK'da kullanılan SQL sorgu mantığının bir benzeri kullanılmıştır. Sonuçlar oluşturulurken bir dakikalık zaman

pencereleri kullanılmıştır. Oluşan sonuçlar daha sonra Apache HBase veri tabanına ithal edilmiştir. Şekil 4.5’de Test ve Doğrulama alt yapısı gösterilmiştir.



Şekil 4.5. Test/Doğrulama alt yapısı

Lamda Mimarinin önemli özelliklerin birisi olan verinin ham halde saklanıyor olmasının avantajı burada kullanılmıştır. Duran hale getirilmiş veriler ham halde HDFS’de saklanmaktadır. HDFS’deki bu duran veriler DVK tarafından belirli zamanlarda işlenmektedir. Duran hale getirilmiş aynı veri, test ve doğrulama sistemi tarafından da işlenerek doğrulama verileri elde edilmektedir.

Bu yazılım Python dilinde geliştirilmiş olup, belirli zaman pencereleri için LM sunum katmanını ve test verilerini sorgulayarak karşılaştırmaktadır. Söz konusu yazılımın çok iş parçacıklı (multi-thread) olacak şekilde paralelleştirilerek geliştirilmesi sonucunda gecikmelerin önüne geçilmiştir. Aynı anda birden fazla iş parçacığı test ve doğrulama işlemini gerçekleştirmektedir. SK’nın tamamının bir defada doğrulanması yaklaşık olarak 10 dakika sürmektedir.

Analitik metrikler daha önce de bahsedildiği gibi test seti için bir dakikalık çözünürlükte hesaplanmıştır. LM sistemi de aynı analitik metrikler üzerine gerçek zamanlı olarak çalışmakta ve sunum katmanında sonuçları hazır hale getirmektedir. Test sistemi bu verileri, farklı teknoloji (Hadoop-Hive) kullanarak ürettiği verilerle karşılaştırmaktadır. Bu şekilde Geri Getirme ve Kesinlik değerleri hesaplanmakta ve görselleştirilmesi için başka bir dosyaya kayıt edilmektedir.

Test/doğrulama alt yapısı LM çalışmaya başladıktan sonra derhal, çok kısa süre aralıkları ile sunum katmanını sorgulamaya başlar ve önceden hesaplanmış olan ilk Geri Getirme ve Kesinlik değerine ulaştığında bir sonraki zaman penceresi için sorgulama işlemine devam eder. Bu işlem ardışık olarak bir sonraki zaman penceresine kayarak devam eder. Bir sonraki zaman penceresinde sürekli ve kısa süreli sorgulamalar yapılmaz, sadece bir önceki zaman penceresi ile tamamlanmakta olan zaman penceresi arasındaki zaman dilimi için sorgulama işlemi gerçekleştirilir. Bu işlemin nedeni sistem test edilirken her seferinde sıfır veri ile başlatılmasıdır (cold start). Dolayısı ile test/doğrulama sistemi ile LM sunum katmanının eşzamanlı hale getirilmesi sağlanmalıdır.

4.3. Tasarlanan Lamda Mimarinin Test Edilmesi

Tasarlanan Lamda Mimarinin deneysel olarak test edilmesi bu bölümde anlatılmıştır. Literatürde açıklanan lamda mimarilerden farklı olarak tamamen özgün olarak tasarlanan ve birçok yeniliği kendi bünyesine katan bu tasarım, öncelikle azami veri işleme kapasitesinin belirlenmesi için test edilmiş, performansı ve güvenilirliği incelenmiştir. Daha sonra normal çalışma durumunda Lamda Mimarinin davranışı araştırılmıştır. Bu çalışma ile Lamda Mimarinin gerçek hayatta en sık karşılaşılan problemler olan tekrarlı veri gelmesi ve geciken veri gelmesi durumundaki davranışı ve performansı da detaylı olarak araştırılmıştır.

Lamda Mimarinin normal çalışması sırasında AVK tarafında meydana gelebilecek hataları düzeltmek amacıyla kullanılan DVK katmanının çalışmadığı durumlarda, SK'nın performansı ve etkinliği incelenerek test edilmiştir. Ayrıca yine DVK çalışmadığında tekrarlı veri ve geciken veri gelmesi durumunda sistemin performansı ve güvenilirliği detaylı şekilde ayrı ayrı test edilmiştir.

Bu çalışmada tasarlanan Lamda Mimarinin toplamda yedi farklı durumda test edilmesi, performansının ve güvenilirliğinin belirlenmesi sağlanmıştır. Gerçekleştirilen yedi farklı testten yalnızca 2 tanesi literatürde yer almakta, 5 tanesi ise ilk defa gerçekleştirilmektedir. Ayrıca yine literatürde yer alan Lamda Mimarilerin sunum katmanları için detaylı tasarımlar yapılmaması nedeniyle, sunum katmanının Lamda Mimarideki etkisinin incelenmesi konusunda yeteri kadar bilgi mevcut değildir. Bu eksiklikleri gidermek için

Lamda Mimari konsepti her yönüyle ele alınarak tasarlanan sistem, gerçek hayatta oluşabilecek en kötü durumlar için test edilmiştir.

Lamda Mimari konsepti baştan sona uygulanarak geliştirilen bu tasarımın detaylı olarak incelenmesi için daha önceki bölümlerde anlatılan test ve doğrulama sistemi ayrıca geliştirilmiştir. Tüm testlere, tasarlanan Lamda Mimari üzerinde hiç veri olmadan (cold start durumunda) başlanmıştır.

Her bir testin aynı koşullarda yapılması gerekmektedir. Aksi durumda testler güvenilir olmayacak ve incelenecek her bir durumun diğerleri ile karşılaştırılması mümkün olmayacaktır. Söz konusu testler daha önce anlatılan test/doğrulama alt yapısı ve donanım kullanılarak gerçekleştirilmiştir. Her bir test için sonuçlar en az üç defa elde edilerek tutarlılıkları incelenmiştir. Gerçekleştirilen testlerde incelen performans ölçütleri geri getirme ve kesinlik değerleri olarak seçilmiştir [30]. Testler esnasında her bir durum incelenirken aynı koşulların gerçekleştiğinden emin olmak için sistem izleme alt yapısı kullanılmıştır. Bu alt yapı sayesinde donanım ve kümelemede oluşan hatalar incelenmiş ve eğer hata varsa test tekrar edilmiştir.

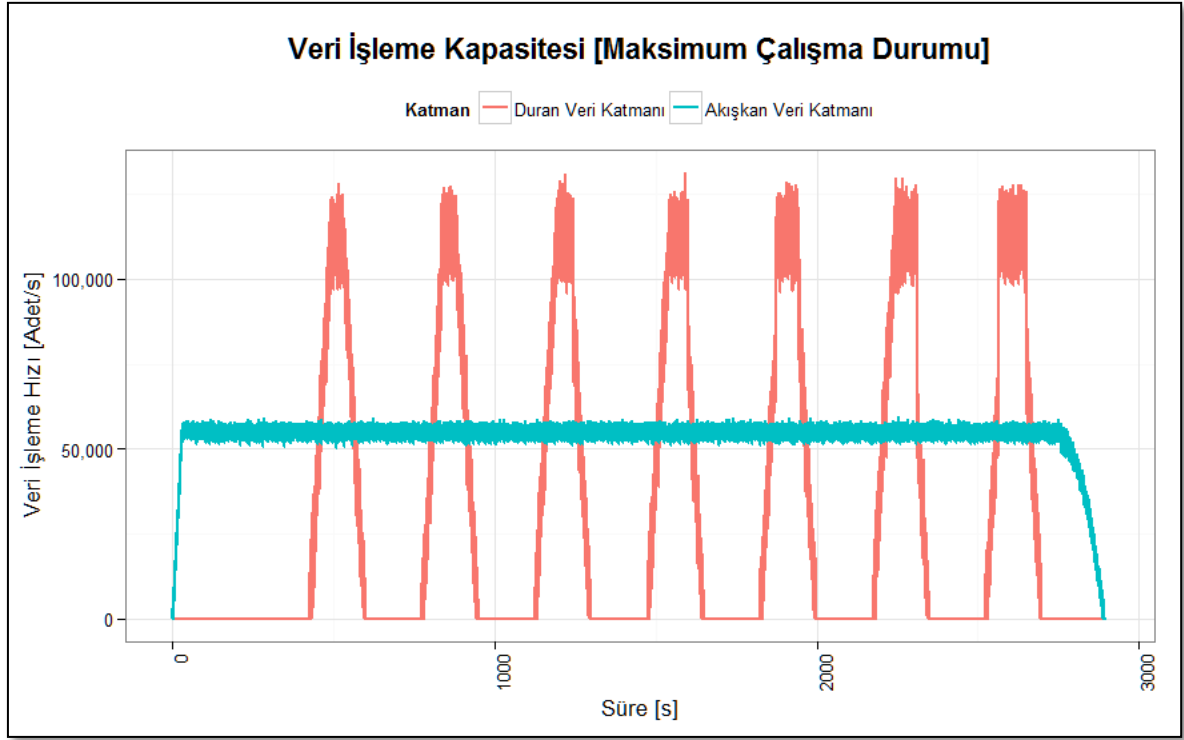
31 Ağustos 2015 ve 1 Ekim 2015 tarihleri arasında her bir anten için toplanan canlı veri hiçbir işleme tabi tutulmadan, en ham haliyle üç farklı dosyaya kayıt edilmiştir. Söz konusu verilerin yazıldığı her bir dosya yaklaşık olarak 66 GB'lık bir veri kümesi oluşturmaktadır. Toplanan veriler içerisinde zamanında gelmeyen ya da tekrarlı gelebilecek veriler olabileceği nedeniyle, test sonuçlarını etkilememesi için Apache Hadoop ve MapReduce tabanlı bir algoritma ile veriler sinyal geliş zamanına göre sıralanmış ve kayıt sayısı bilgisi hesaplatılmıştır. Test sonuçları, olması gereken değerler kullanılarak kontrol edilirken, geciken ve tekrarlı gelen verilerin testi etkilemesinin önüne bu şekilde geçilebilmiştir. Test verileri, ilerleyen bölümlerde açıklanacağı üzere veri yolu üzerinde kontrollü bir şekilde geciktirilerek veya tekrarlı veri gönderilerek, sistemin davranışı güvenilir bir şekilde gözlemlenebilecektir. Test yapılacak veri setinin elde edilmesi için son olarak yine MapReduce tabanlı bir algoritma ile 1-30 Eylül 2015 tarih aralığındaki veriler süzülüş ve testlerde performans ölçütlerinin hesaplanması için kullanılmıştır. Tasarlanan ve test edilecek LM'nin analiz sonucunda oluşturması istenen analitik metrikler aşağıdaki gibidir:

- Ortalama Tekil Uçak Sayısı
- Ortalama Tekil Bilinmeyen Uçak Sayısı
- Minimum Hava Taşıtı Hızı
- Maksimum Hava Taşıtı Hızı
- Minimum Hava Taşıtı Mesafesi
- Maksimum Hava Taşıtı Mesafesi
- Minimum İrtifa
- Maksimum İrtifa

AVK'nın çalışmadığı ancak DVK'nın çalıştığı durum ayrıca incelenmemiştir. Bu tip bir durumun gerçekleşmesi halinde gelen verilerin işlenmesi sadece DVK tarafından gerçekleştirilecektir. Dolayısı ile lamda mimari konsepti doğrultusunda bir çalışma değil tamamen duran veri analizi gerçekleştirilmiş olacağı için bu durum ayrıca incelenmemiştir. Ancak DVK'nın çalışmadığı ve AVK'nın çalıştığı durum da sadece akışkan veri analizi olarak nitelendirilebilir. Bu durum da lamda mimari konsepti doğrultusunda bir çalışma olmamaktadır fakat DVK'nın düzeltici etkisinin net bir şekilde ortaya konabilmesi için bu durum özellikle incelenmiştir.

4.3.1. Azami kapasitede çalışma testi

Azami kapasite ile çalışma testinde sistemin veri işleme kapasitesinin limitleri ölçülmüştür. Bu testte kesinlik ve geri getirme değerlerinin 1'e eşit olduğu yani %100 olduğu durumda Lamda Mimarideki maksimum veri akış hızı ölçülmektedir. Şekil 4.6'da test süresince veri işleme hızının zamana göre değişimi gösterilmiştir.

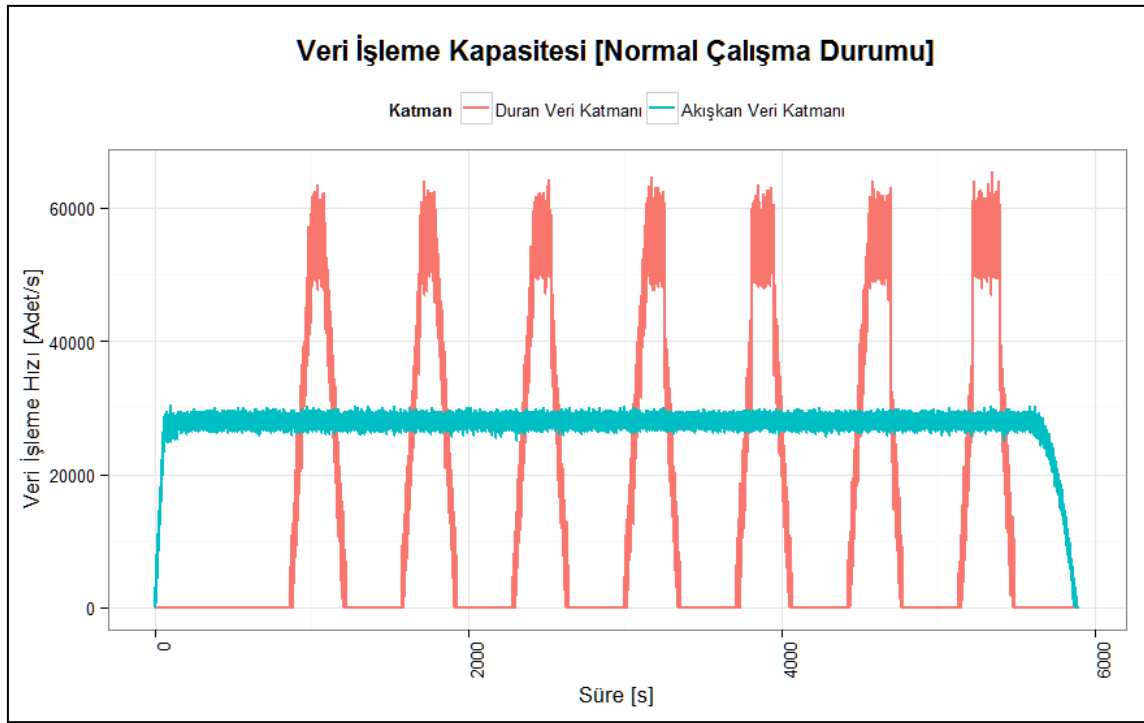


Şekil 4.6. Performans testi sonucu veri işleme hızının zamana göre değişimi

Bu testte veri yolundan gelen verilerin işlenmesi toplam 2902 saniyede tamamlanmıştır. Bu süre zarfında AVK'da maksimum veri işleme hızının ortalama 52000 adet/s olduğu belirlenmiştir. DVK'da ise maksimum veri işleme hızı ortalama 130 000 adet/s olarak tespit edilmiştir. DVK'nın verileri iki kat daha hızlı işleyebildiği ve test süresince, tüm zaman aralıklarında veri işlemede AVK'dan geride kalmadığı yani gecikmediği görülmüştür. Bu test gerçekleştirilirken veri işleme kapasitesinin yetmediği durumlarda bazı aksaklıklar yaşanmış ve bu aksaklıklar sistem izleme alt yapısı ile tespit edildikten sonra test üç defa daha tekrar edilmiştir. Üç tekrardan sonra sonuçlar karşılaştırılarak tutarlı olduğu belirlenmiştir.

4.3.2. Normal kapasite çalışma testi

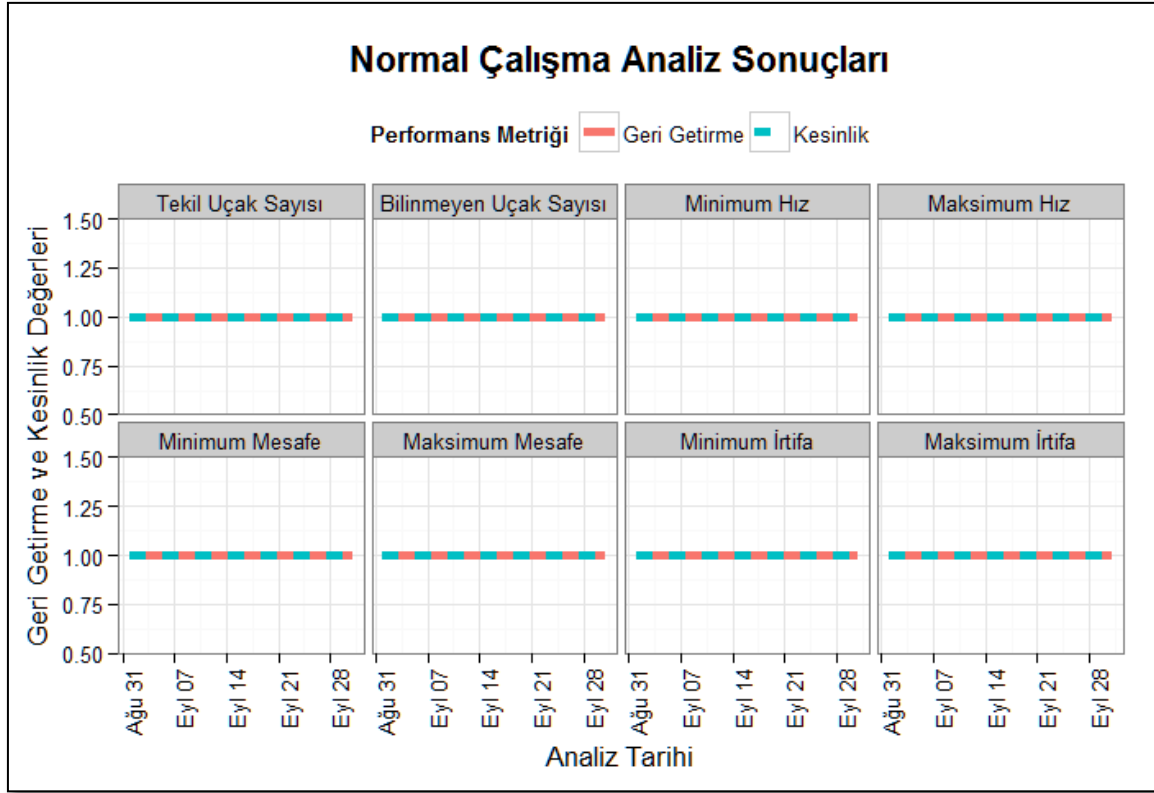
Normal kapasite çalışma testinde Lamda Mimari çalışması gereken nominal durumda incelenmiştir. Veri yolundan alınan veriler her iki katmanda da işlenerek sunum katmanına iletilmektedir. Şekil 4.7'de normal çalışma durumunda veri işleme hızının zamana göre değişimi gösterilmiştir.



Şekil 4.7. Normal çalışma durumunda veri işleme hızının zamana göre değişimi

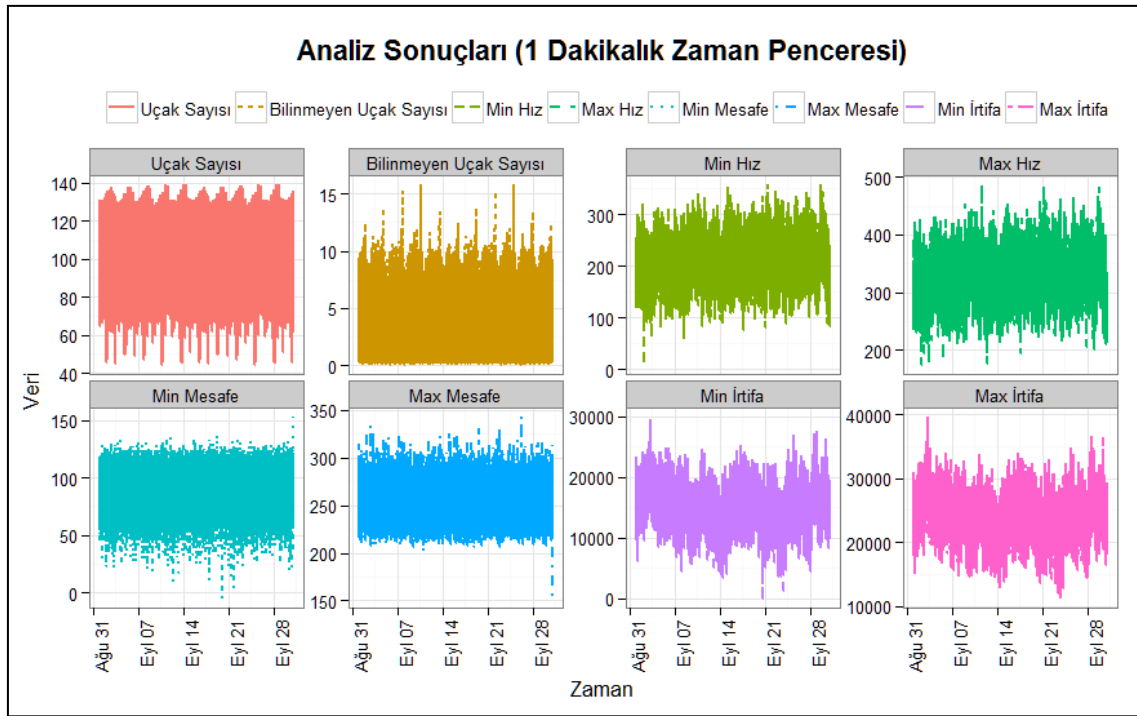
Lamda Mimarinin tüm veriyi işlemesinin yaklaşık olarak 5902 saniye sürdüğü tespit edilmiştir. Bu süre zarfında DVK'nın veri işleme hızının ortalama 56000 adet/s olduğu AVK'nın veri işleme hızının ise ortalama 27000 adet/s olduğu görülmüştür. DVK'nın minimum çalışma süresi 352 saniye, maksimum çalışma süresi ise 362 saniye olarak belirlenmiştir. Test süresince DVK'nın veri işlemede AVK'dan geride kalmadığı yani gecikmediği gözlemlenmiştir. Bu test gerçekleştirilirken Lamda Mimaride herhangi bir hata oluşmadığı sistem izleme alt yapısı ile tespit edilmiştir. Test üç defa tekrarlanarak sonuçların tutarlı olduğu belirlenmiştir.

Performans ölçütleri olan kesinlik ve geri getirmenin analiz tarihine (01-30 Eylül 2015) göre değişimi Şekil 4.8'de sunulmuştur. Analiz bir dakikalık zaman penceresi aralıkları ile gerçekleştirilmiştir. Toplam 43200 zaman penceresinde kesinlik ve geri getirme değerlerinin 1'e eşit olduğu ve değişmediği, bu noktadan hareketle, zaman pencerelerinin hiç birinde Lamda Mimarinin hata yapmadığı belirlenmiştir. Sistemin normal çalışma durumunda oldukça güvenilir sonuçlar verdiği tespit edilmiştir.

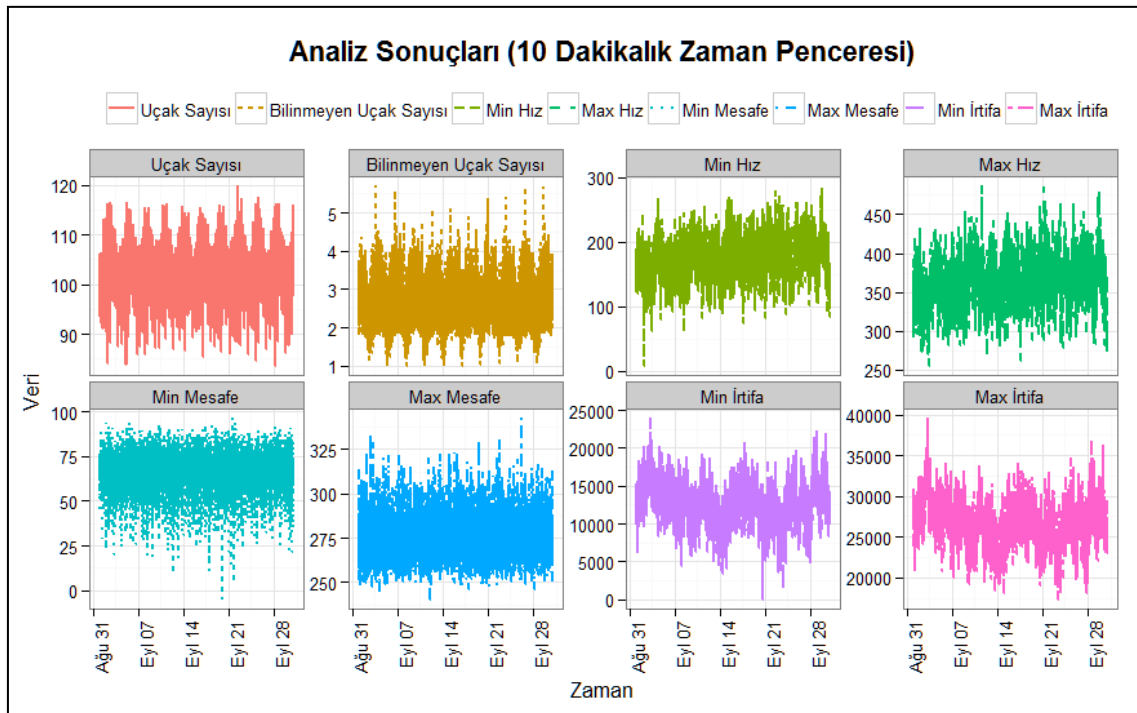


Şekil 4.8. Normal çalışma durumu geri getirme ve kesinlik değerlerinin değişimi

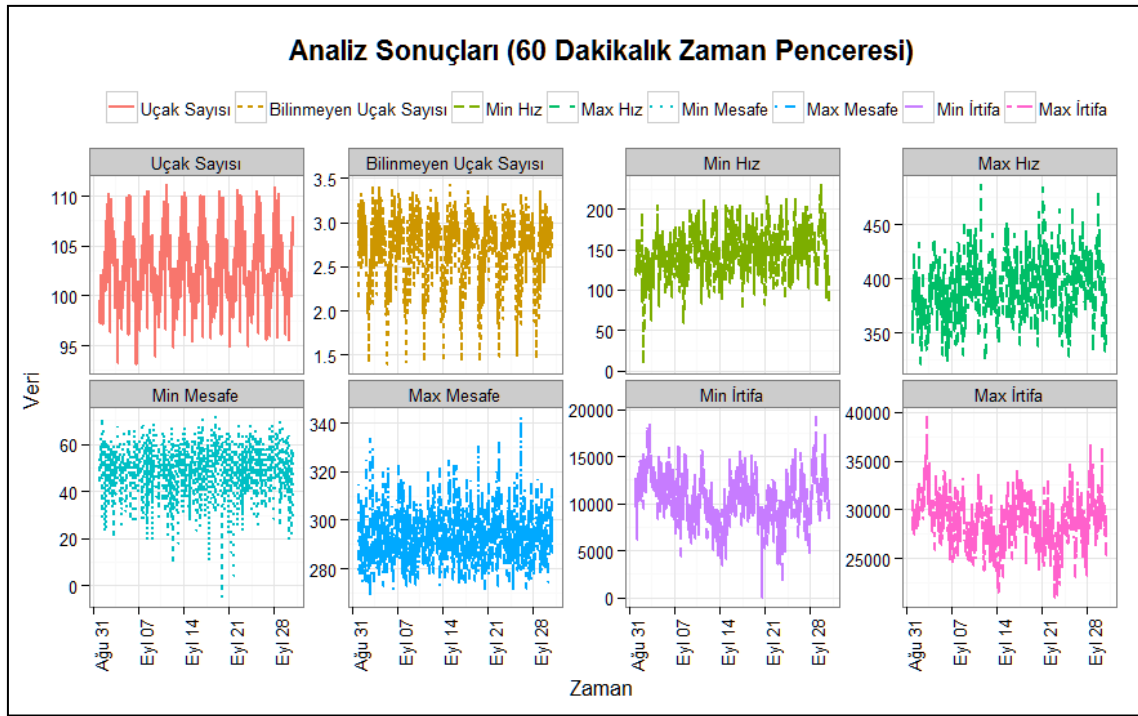
01-30 Eylül 2015 tarihleri arasındaki veriler Lamda Mimaride işlenerek, ortalama tekil uçak sayısı, ortalama tekil bilinmeyen uçak sayısı, minimum hava taşıtı hızı, maksimum hava taşıtı hızı, minimum hava taşıtı mesafesi, maksimum hava taşıtı mesafesi, minimum irtifa ve maksimum irtifa metrikleri analiz sonuçları 1, 10, 60 dakikalık zaman pencereleri için elde edilmiştir. Analiz sonuçları 1 dakikalık zaman penceresi için Şekil 4.9’da, 10 dakikalık zaman penceresi için Şekil 4.10’da ve 60 dakikalık zaman penceresi için Şekil 4.11’de gösterilmiştir.



Şekil 4.9. 01-30 Eylül 2015 tarihleri arası 1 dk'lık zaman penceresi için analiz sonuçları



Şekil 4.10. 01-30 Eylül 2015 tarihleri arası 10 dk'lık zaman penceresi için analiz sonuçları



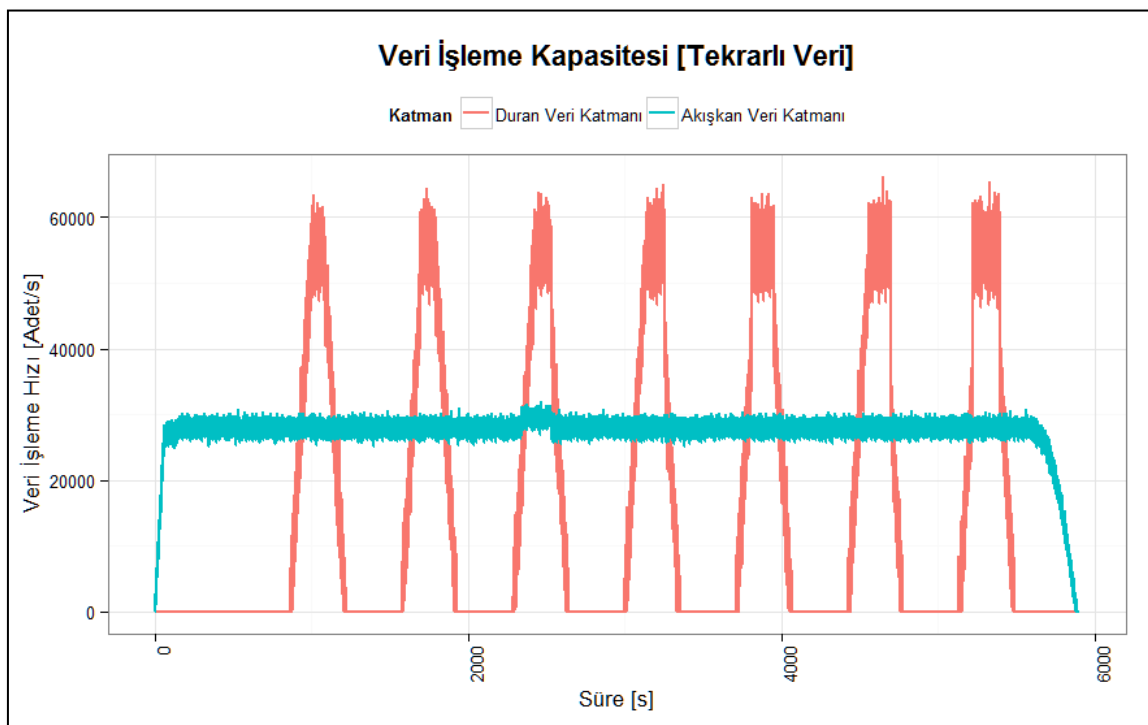
Şekil 4.11. 01-30 Eylül 2015 tarihleri arası 1 saatlik zaman penceresi için analiz sonuçları

Farklı zaman pencereleri için gerçekleştirilen analizler sonucunda 1 aylık süreye ait verilerin hızlı bir şekilde ve yaklaşık olarak 0,456 – 0,584 saniyede sorgulanabildiği görülmüştür. Yeri veri gelişi ve LM tarafından işlenmesi ile bu süre çok fazla değişmemektedir. Daha düşük zaman çözünürlüğünde ithal edilen verilerin işlenmesinin, kümeleme kaynaklarını daha fazla tükettiği tespit edilmiştir.

4.3.3. Tekrarlı veri gelmesi ve Lamda Mimarinin davranışının incelenmesi

Tekrarlı veri gelmesi testinde Lamda Mimari çalışması gereken nominal durumda incelenmiştir. Veri yolundan alınan veriler her iki katmanda da işlenerek sunum katmanına iletilmektedir. Şekil 4.12’de tekrarlı veri gelmesi durumunda veri işleme hızının zamana göre değişimi gösterilmiştir. Bu testte, 15 Eylül 2015 saat 10:00 ile 16 Eylül 2015 saat 16:00 tarihleri arasındaki veriler, DVK’nın üçüncü defa çalışmaya başladığı anda tekrarlanacak şekilde veri yolundan gönderilmektedir. DVK çalışmaya başladığı anda bu verilerin tekrarlı olarak gelmesi yaşanacak en kötü durumlardan birisidir. DVK’nın tekrarlı olarak gelen veriyi düzeltmesi ancak bir sonraki DVK çalışmasında gerçekleştirilebilecektir. Bu nedenle bu anormal durumun incelenmesi ile Lamda Mimari davranışı test edilmiştir.

Lamda Mimarinin tüm veriyi işleminin yaklaşık olarak 5930 saniye sürdüğü tespit edilmiştir. Bu süre zarfında DVK'nın veri işleme hızının ortalama 57124 adet/s olduğu AVK'nın veri işleme hızının ise ortalama 26928 adet/s olduğu görülmüştür. DVK'nın minimum çalışma süresinin 353 saniye, maksimum çalışma süresinin ise 361 saniye olduğu belirlenmiştir. Test süresince DVK'nın veri işlemede AVK'dan geride kalmadığı yani gecikmediği gözlemlenmiştir. Bu test gerçekleştirilirken Lamda Mimaride herhangi bir hata oluşmadığı sistem izleme alt yapısı ile tespit edilmiştir. Test üç defa tekrarlanarak sonuçların tutarlı olduğu belirlenmiştir.



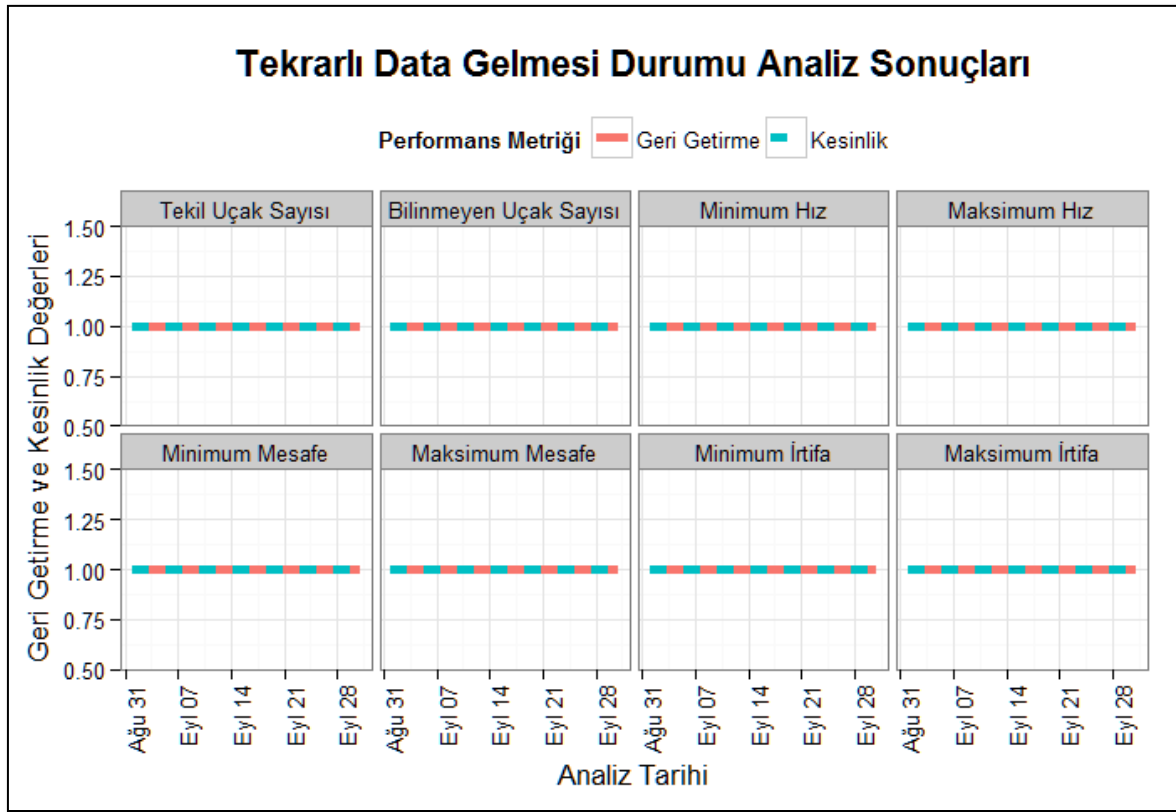
Şekil 4.12. Tekrarlı veri gelmesi durumunda veri işleme hızının zamana göre değişimi

Performans ölçütleri olan kesinlik ve geri getirmenin analiz tarihine (01-30 Eylül 2015) göre değişimi Şekil 4.13'de gösterilmiştir. Analiz bir dakikalık zaman penceresi aralıkları ile gerçekleştirilmiştir. Toplam 43200 zaman penceresinde kesinlik ve geri getirme değerlerinin değişmediği, 1'e eşit olduğu bu noktadan hareketle zaman pencerelerinin hiçbirinde Lamda Mimarinin hata yapmadığı belirlenmiştir. Sistemin normal çalışma durumunda oldukça güvenilir sonuçlar verdiği ve hata yapmadığı tespit edilmiştir.

Daha önceden de bahsedildiği gibi, AVK ve DVK'da verinin işlenmesi aynı yöntem ile yapılmaktadır. Başka bir deyişle bir defa kod geliştirilmekte ve bu kod iki katmanda da

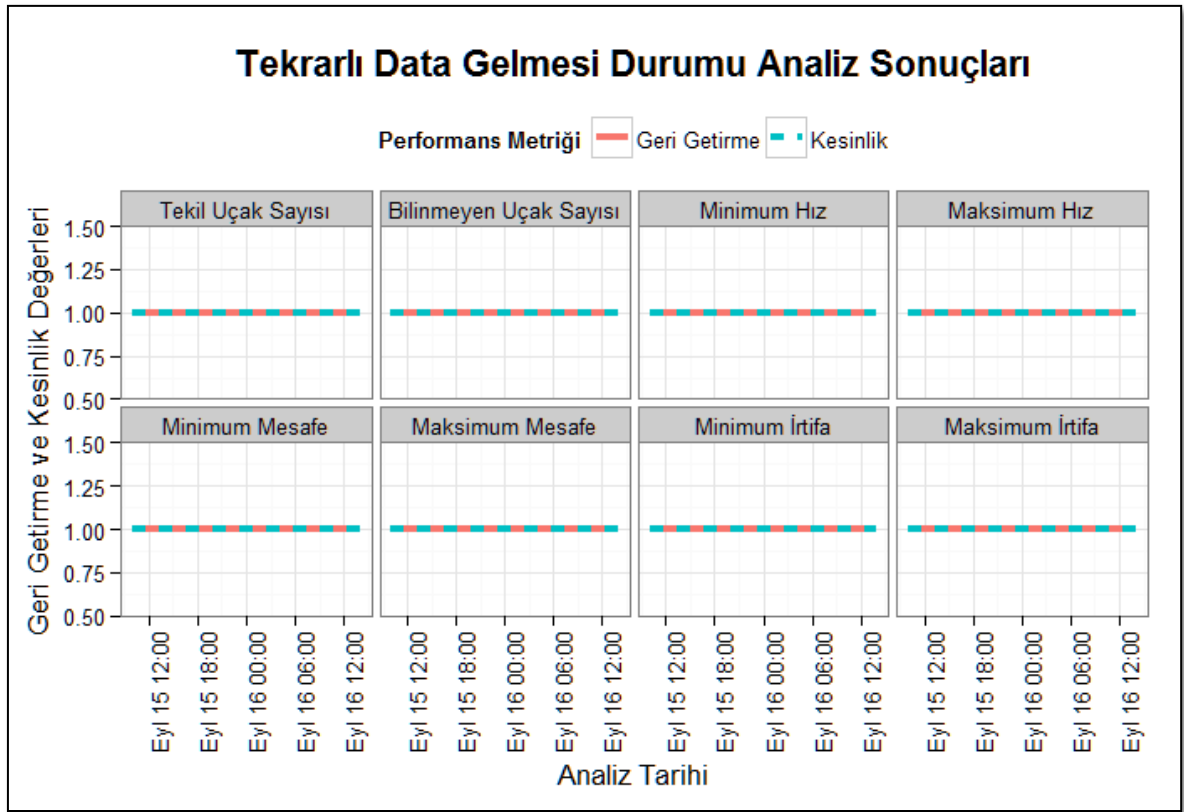
kullanılmaktadır. Birden fazla kaynaktan verinin geliyor olması ile bir verinin birçok defa tekrarlanması arasında, işlenmesi açısından bir fark yoktur. Üç farklı kaynaktan verinin zaman farkı ile gelen verilerin tek bir veri haline dönüştürülmesi işleminin geliştirilen algoritma ile yapıldığı daha önce de belirtilmiştir. Analiz yöntemi yani algoritma geliştirilirken bu husus dikkate alınmış ve zaman pencereleri içerisinde indirgeme işlemi uygulanmıştır. Bu sayede her iki katmanda da sonuçlarda hata tespit edilmemiştir.

Bu hususun dikkate alınmadığı durumlarda, elde edilecek sonuçların geri getirme değerinin artması, ancak kesinlik değerinin de azalması şeklinde olacağı öngörülmektedir. AVK tarafından bu şekilde SK'ya iletilen veriler, normalden fazla verinin sorgu sonucunda döndürülmesini sağlamayacaktır. Bunun nedeni daha önce tasarım bölümlerinde de anlatıldığı gibi SK'ya verinin belirli bir şema ile ithal edilmesi, bu şemada belirtilen işlemlerin yapılıyor olmasıdır. Şemada belirtilen tekillik kestirimi (cardinality estimation) ve minimum-maksimum işlemleri, ithal edilen veriye gerçek zamanlı olarak, zaman penceresi tamamlanana kadar uygulanmaktadır. Veri tekrarlı gelse bile buradaki işlemlerden ötürü tekillik kestirimi aynı sonucu verecektir. Yine, veri tekrarlı geldiğinde minimum ve maksimum alma işlemleri sayesinde aynı sonuçlar elde edilecektir. Aynı veriden binlerce sayıda geldiğinde bu tekrarlı verilerin minimum ya da maksimumu bir defa geldiği zamanki minimum ve maksimum ile aynı olacaktır. Dolayısı ile gerçekleştirilen tasarım sayesinde SK'da ikinci bir emniyet önlemi olarak şema tabanlı veri ithali kullanılmıştır. AVK katmanında ETL ve diğer işlemlerde bir sorun yaşanmadığı sürece tekrarlı olarak gelen veriler SK'daki geri getirme ve kesinlik değerlerini etkilememektedir.



Şekil 4.13. Tekrarlı veri gelme durumu geri getirme ve kesinlik değerlerinin değişimi

Veri işleme algoritmasının tasarımı esnasında bu durum önceden dikkate alındığı için bir defa geliştirilen kodun her iki katmanda da kullanılması sayesinde SK'daki sorgularda hata oluşmamıştır. 15 Eylül 2015 saat 10:00 ile 16 Eylül 2015 saat 16:00 tarihleri arasında tekrarlı olarak gönderilen verilerin işlenmesi ile elde edilen analiz sonucu, Şekil 4.14'de bu tarih aralığı için daha yakından gösterilmektedir.



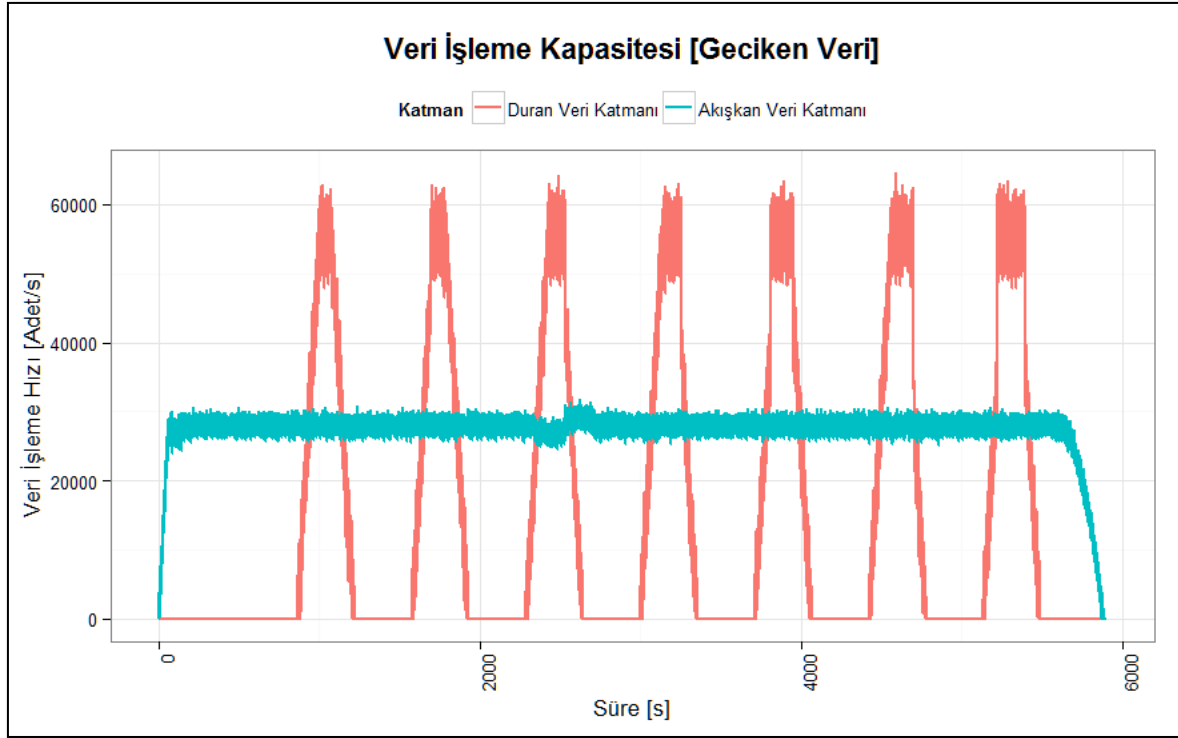
Şekil 4.14. Tekrarlı veri gelme durumu geri getirme ve kesinlik değerlerinin değişimi

4.3.4. Veri gecikmesi ve lamda mimarinin davranışının incelenmesi

Veri gecikmesi testinde, Lamda Mimari, çalışması gereken nominal durumda incelenmiştir. Veri yolundan alınan veriler her iki katmanda da işlenerek sunum katmanına iletilmektedir. Şekil 4.15’de veri gecikmesi durumunda veri işleme hızının zamana göre değişimi gösterilmiştir. Bu testte, 15 Eylül 2015 saat 10:00 ile 16 Eylül 2015 saat 14:00 tarihleri arasındaki veriler DVK’nın üçüncü defa çalışmaya başladığı anda geciktirilerek, daha sonra iletilecek şekilde veri yolundan gönderilmektedir. DVK çalışmaya başladığı anda bu verilerin geç gelmesi yaşanacak en kötü durumdur. DVK’nın sorunlu gelen veriyi düzeltmesi ancak bir sonraki DVK çalışmasında gerçekleştirilebilecektir. Bu nedenle en kötü durumun incelenmesi ile Lamda Mimari test edilmiştir.

Lamda Mimarinin tüm veriyi işleme hızı yaklaşık olarak 5925 saniye sürdüğü tespit edilmiştir. Bu süre zarfında DVK’nın veri işleme hızının ortalama 55214 adet/s olduğu, AVK’nın veri işleme hızının ise 27111 adet/s olduğu görülmüştür. DVK’nın minimum çalışma süresinin 350 saniye, maksimum çalışma süresinin ise 365 saniye olduğu

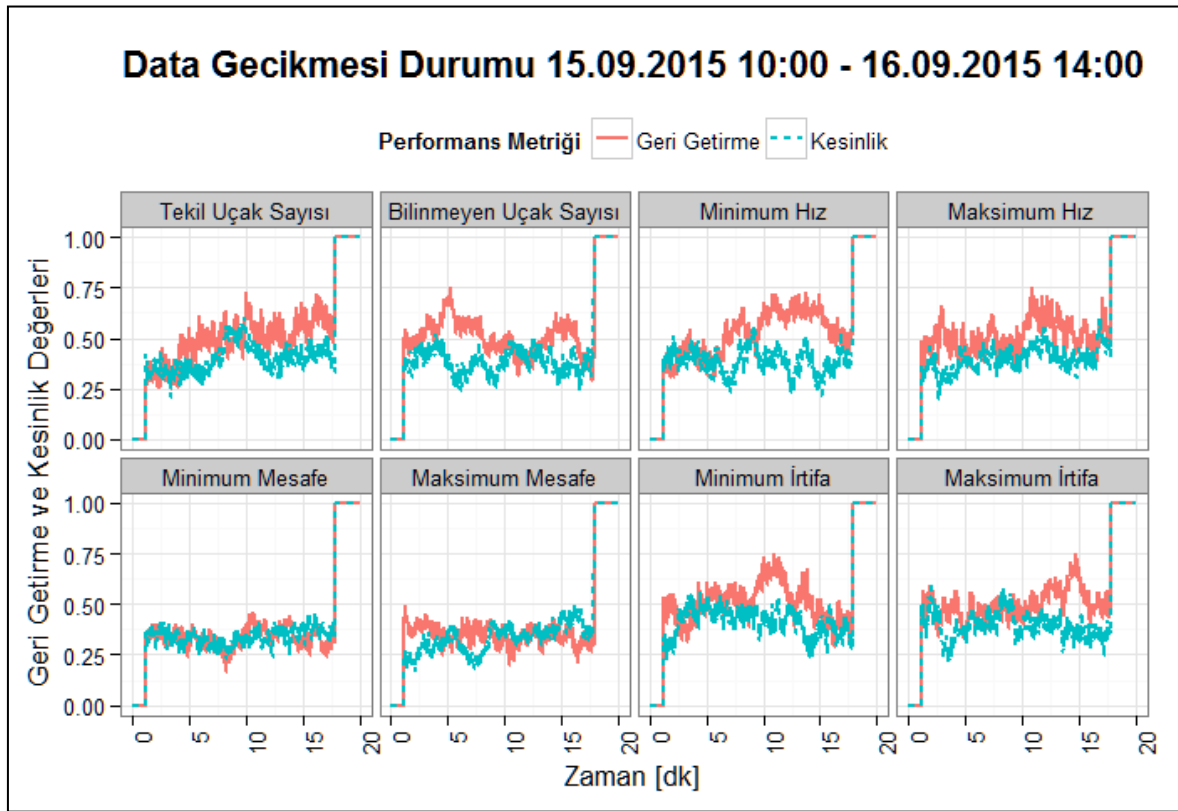
belirlenmiştir. Test süresince DVK'nın veri işlemede AVK'dan geride kalmadığı yani gecikmediği gözlemlenmiştir. Bu test gerçekleştirilirken Lamda Mimaride herhangi bir hata oluşmadığı sistem izleme alt yapısı ile tespit edilmiştir. Test üç defa tekrarlanarak sonuçların tutarlılığı kontrol edilmiştir.



Şekil 4.15. Verinin gecikmesi durumunda veri işleme hızının zamana göre değişimi

Daha öncede belirtildiği gibi sistem olabilecek en kötü durumda test edilmiştir. Şekil 4.16'da veri gecikmesi durumunda geri getirme ve kesinlik değerlerinin zamana göre değişimi gösterilmektedir. Söz konusu şekilde 15 Eylül 2015 saat 10:00 ile 16 Eylül 2015 saat 14:00 arasındaki kayıtların sorgulanması ile elde edilen ortalama kesinlik ve geri getirme değerlerinin 20 dakika boyunca değişimi gösterilmektedir. Belirli bir süre geri getirme ve kesinlik değerleri sıfır olmakta, daha sonra verilerin işlenmesi ile bu değerler değişmektedir. Bu değişim hiçbir zaman 1 değerine ulaşamamaktadır. Bunun nedeni daha önceden de bahsedildiği gibi SK'daki veri reddetme politikasının geciken verileri işlemeyecek şekilde ayarlanmasıdır. Lamda Mimarinin çalışma yönteminde de belirtildiği üzere, veriler gelmesi gereken zamandan sonra geldiği için işlenerek SK'ya iletildiğinde SK bu verileri ithal etmemektedir. Yine Lamda Mimarinin çalışma yönteminde belirtildiği üzere, veriler geç gelse bile DVK'da daha sonra işlenmesi için ham haliyle biriktirilmektedir. Daha sonraki bir zamanda DVK'nın çalışmasıyla birlikte biriktirilen

veriler işlenmektedir. Bu çalışmanın sonucunda geri getirme ve kesinlik değerleri tekrar 1'e ulaşmaktadır. Buradan anlaşıldığı üzere tasarlanan lamda mimari, sorunsuz şekilde, çalışması gerektiği gibi çalışmıştır. Lamda Mimarinin eninde sonunda güvenilir bir sistem olduğu (eventually accurate) daha önceki bölümlerde de birçok defa belirtilmiştir. AVK'da oluşan bir sorunun en geç bir sonraki DVK çalışmasına kadar devam ettiği, DVK'nın çalışması tamamlandığında SK'dan yapılan sorgulamalar sonucunda elde edilen verilerin hatasız olduğu görülmektedir.



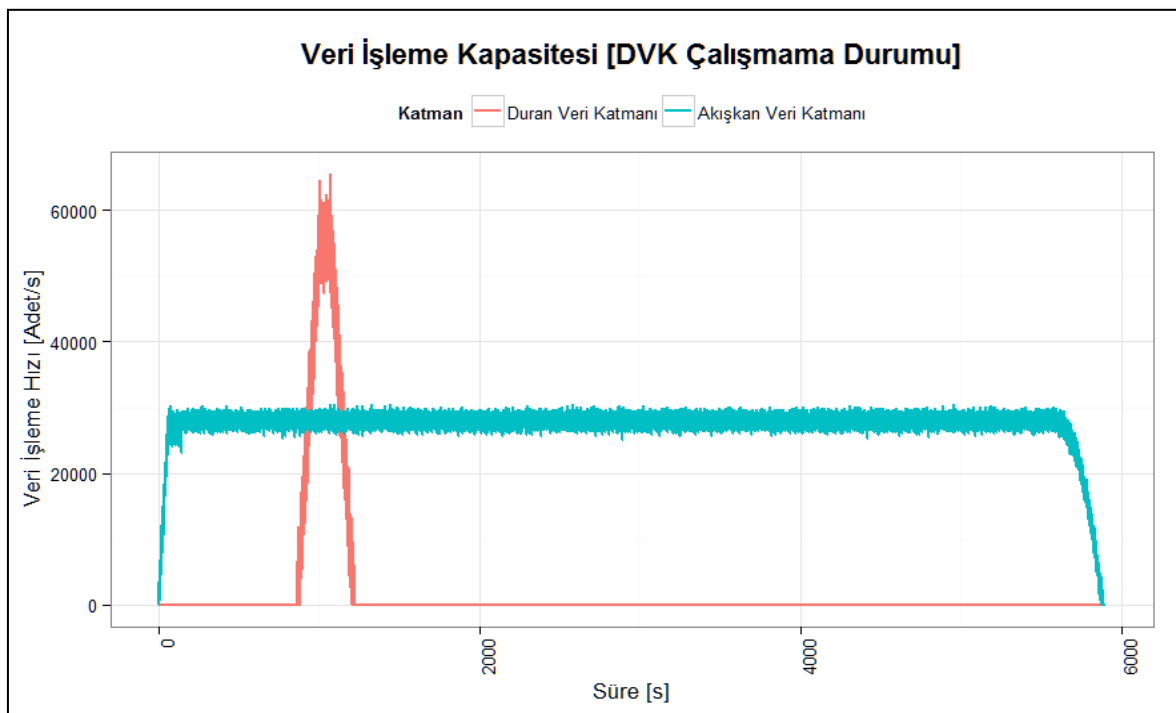
Şekil 4.16. Veri gecikmesi durumunda geri getirme ve kesinlik değerlerinin 15 Eylül 2015 10:00 ile 16 Eylül 2015 14:00 zaman aralığı için değişimi

Şekil 4.16'da görüldüğü üzere geri getirme ve kesinlik değerleri en çok minimum mesafe ve maksimum mesafe verileri üzerinde hataya neden olmuştur. Bu verilerde, geri getirme ve kesinlik değerleri oldukça düşük seviyelerde kalmaktadır. Daha önceden de belirtildiği gibi Lamda Mimaride işlenmesi en zor metrik minimum ve maksimum mesafedir. Karmaşık hesaplamalar gerektiren bu iki metrik, Lamda Mimaride işlendiğinde AVK'nın daha fazla hataya yapmasına ve SK'da geri getirme ve kesinlik değerlerinin düşmesine neden olmaktadır. Karmaşık hesaplama işlemlerinin bu gibi durumlarda hataya karşı oldukça hassas olduğu görülmektedir.

4.3.5. Duran veri katmanının çalışmaması durumunda Lamda Mimari davranışının İncelenmesi

Duran veri katmanının çalışmaması durumunda Lamda Mimari davranışı, nominal çalışma koşullarında incelenmiştir. Veri yolundan alınan veriler her iki katmanda da işlenerek sunum katmanına iletilmektedir. Şekil 4.17’de DVK’nın çalışmadığı durumda veri işleme hızının zamana göre değişimi gösterilmiştir. Bu testte, DVK’nın sadece bir defa çalıştırıldıktan sonra bir daha çalışmasına engel olunarak Lamda Mimarinin tepkisi incelenmiştir. Bu durumda AVK’ya gelen veriler işlenerek SK’da kullanıma sunulacaktır. Sistemin bu durumdaki başarısı ve güvenilirliği sadece AVK’nın başarısına bağlıdır.

Lamda Mimarinin tüm veriyi işlemesi sadece AVK ile yaklaşık olarak 5910 saniye sürdüğü tespit edilmiştir. Bu süre zarfında DVK’nın veri işleme hızının sıfır olduğu AVK’nın veri işleme hızının ise ortalama 28558 adet/s olduğu görülmüştür. DVK’nın bir defa çalıştırıldığında çalışma süresinin 345 saniye olduğu belirlenmiştir. Bu test gerçekleştirilirken akışkan veri katmanında herhangi bir hata oluşmadığı sistem izleme alt yapısı ile tespit edilmiştir. Test bu defa beş kez tekrarlanarak sonuçların tutarlılığı kontrol edilmiştir.

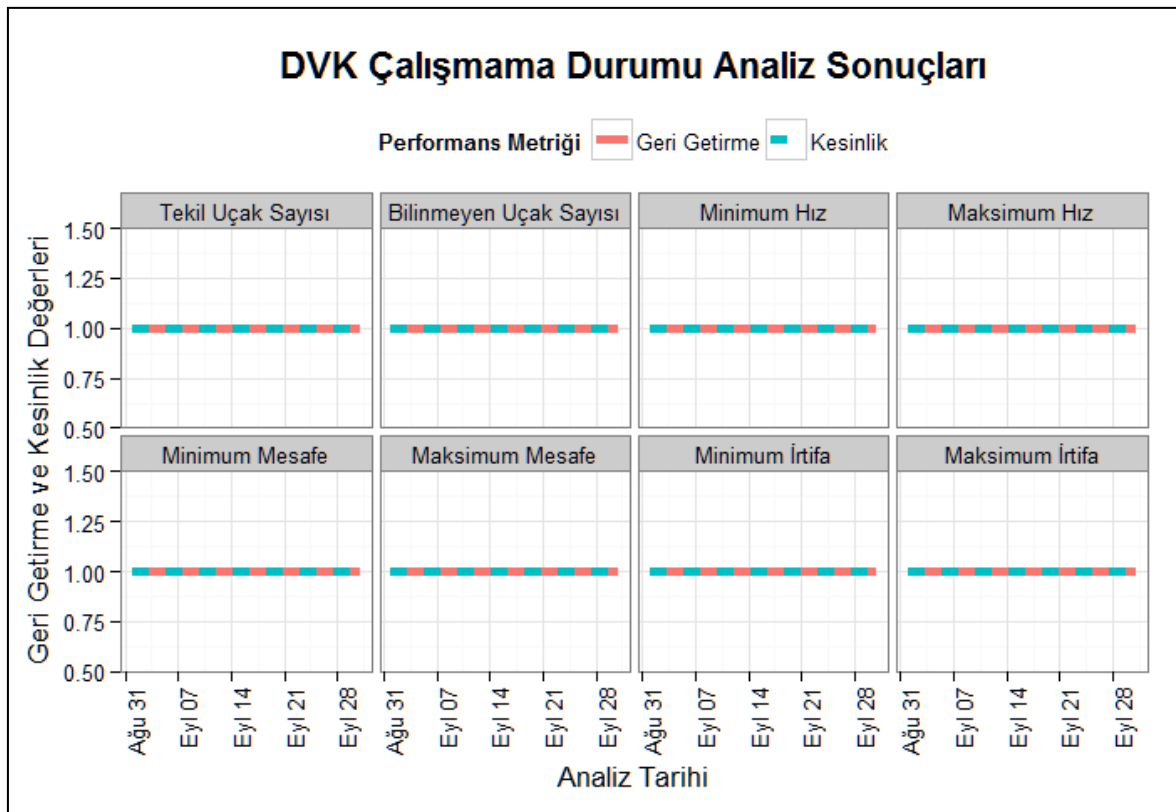


Şekil 4.17. DVK çalışmadığı durumda veri işleme hızının zamana göre değişimi

Performans ölçütleri olan kesinlik ve geri getirmenin analiz tarihine (01-30 Eylül 2015) göre değişimi Şekil 4.18'de gösterilmiştir. Analiz bir dakikalık zaman penceresi aralıkları ile gerçekleştirilmiştir. Toplam 43200 zaman penceresinde kesinlik ve geri getirme değerlerinin değişmediği, 1'e eşit olduğu, bu noktadan hareketle zaman pencerelerinin hiç birisinde Lamda Mimarinin hata yapmadığı belirlenmiştir. Sistemin DVK'nın çalışmadığı durumda bile oldukça güvenilir sonuçlar verdiği tespit edilmiştir.

Bu test sonucunda görüldüğü gibi, AVK'da yaşanan bir sorun olmadığı ve verinin gecikmesi ya da tekrarlı olarak gelmesi gibi bir durum oluşmadığı için SK'da yapılan sorgulamaların sonucunda geri getirme ve kesinlik değerlerinde bir sorun olmadığı gözlenmektedir. Lamda Mimarinin maksimum performans limitlerinin tespiti için yapılan çalışmada, aşırı yük altında başarılı şekilde çalıştığı gözlenmiştir. Tasarımı daha ağır yük altında çalışacağı durumlara göre yapıldığı için, DVK'nın çalışmaması durumunda herhangi bir sorun yaşanmamıştır. Ayrıca, DVK, AVK ve SK'nın birbirinden ayrı şekilde çalışması başka bir deyişle her katmanın sadece kendisi ile ilgili ve uzman olduğu veri türü için çalışıyor olmasından ötürü, DVK çalışmadığı ve gelen veride gecikme ya da tekrar olmadığı bu durum için sistemin güvenilir olarak çalıştığı değerlendirilmektedir. AVK ve DVK gelen veriyi işleyerek doğrudan SK'ya aktarmaktadır. SK'nın bu testteki veri beslemesi tamamen AVK üzerinden gerçekleşmiştir. Bu nedenle veri iletiminde ve ithalinde sorun yaşanmamasından ötürü bir katmanı çalışmayan lamda mimarinin tüm test sürecinde hatasız ve güvenilir şekilde çalıştığı belirlenmiştir. DVK'nın çalışmadığı durumda veri akışında oluşabilecek sorunlardan dolayı lamda mimarinin ne şekilde davranacağı müteakip maddelerde incelenecektir.

AVK'nın çalışmadığı ancak DVK'nın çalıştığı durum ayrıca incelenmemiştir. Bu tip bir durumun gerçekleşmesi koşulunda, gelen verilerin işlenmesi sadece DVK tarafından gerçekleştirilecektir. Dolayısı ile Lamda Mimari konsepti doğrultusunda bir çalışma değil, tamamen duran veri analizi gerçekleştirilmiş olacağı için, bu durumun ayrıca incelenmesine gerek duyulmamıştır. Ancak DVK'nın çalışmadığı ve AVK'nın çalıştığı durum, sadece akışkan veri analizi olarak nitelendirilebilir. Bu durumda Lamda Mimari konsepti doğrultusunda bir çalışma olmamaktadır fakat DVK'nın düzeltici etkisinin net bir şekilde ortaya konabilmesi için, bu çalışma koşulu özellikle incelenmiştir.



Şekil 4.18. DVK çalışmadığı durumda veri işleme hızının zamana göre değişimi

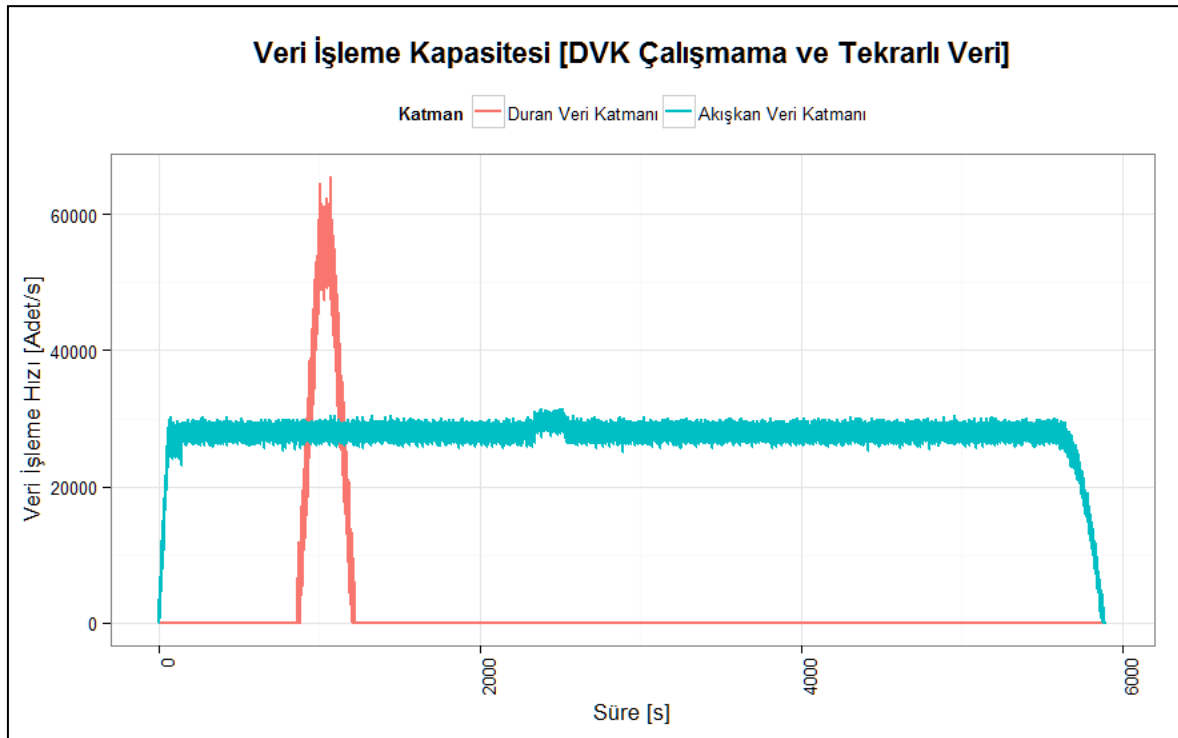
4.3.6. Duran veri katmanının çalışmaması ve tekrarlı veri gelmesi durumunda Lamda Mimari davranışının incelenmesi

Duran veri katmanının çalışmaması ve tekrarlı veri gelmesi halinde Lamda Mimari davranışı nominal durumda incelenmiştir. Veri yolundan alınan veriler her iki katmanda da işlenerek sunum katmanına iletilmektedir. Şekil 4.19'da DVK'nın çalışmadığı durumda veri işleme hızının zamana göre değişimi gösterilmiştir. Bu testte, DVK sadece bir defa çalıştırıldıktan sonra bir daha çalıştırılmamış ve sisteme tekrarlı verinin gönderilmesinden sonra Lamda Mimarinin davranışı incelenmiştir. Bu durumda AVK'ya gelen veriler işlenerek SK'ya iletilmekte ve ithal edilmektedir. Sistemin bu durumdaki başarısı ve güvenilirliği sadece AVK'nın başarısına bağlıdır. Bu testte, 15 Eylül 2015 saat 10:00 ile 16 Eylül 2015 saat 14:00 tarihleri arasındaki veriler DVK'nın bir defa çalışmasından sonra veri yolundan tekrarlı olarak gönderilmektedir.

Lamda Mimarinin tüm veriyi işlemesinin sadece AVK ile yaklaşık olarak 5960 saniye sürdüğü tespit edilmiştir. Bu süre zarfında DVK'nın veri işleme hızının sıfır olduğu AVK'nın veri işleme hızının ise ortalama 29225 adet/s olduğu görülmüştür. DVK bir defa

çalıştırıldığında çalışma süresinin 345 saniye olduğu belirlenmiştir. Bu test gerçekleştirilirken akışkan veri katmanında herhangi bir hata oluşmadığı, sistem izleme alt yapısı ile tespit edilmiştir. Test bu defa beş kez tekrarlanarak sonuçların tutarlılığı kontrol edilmiştir.

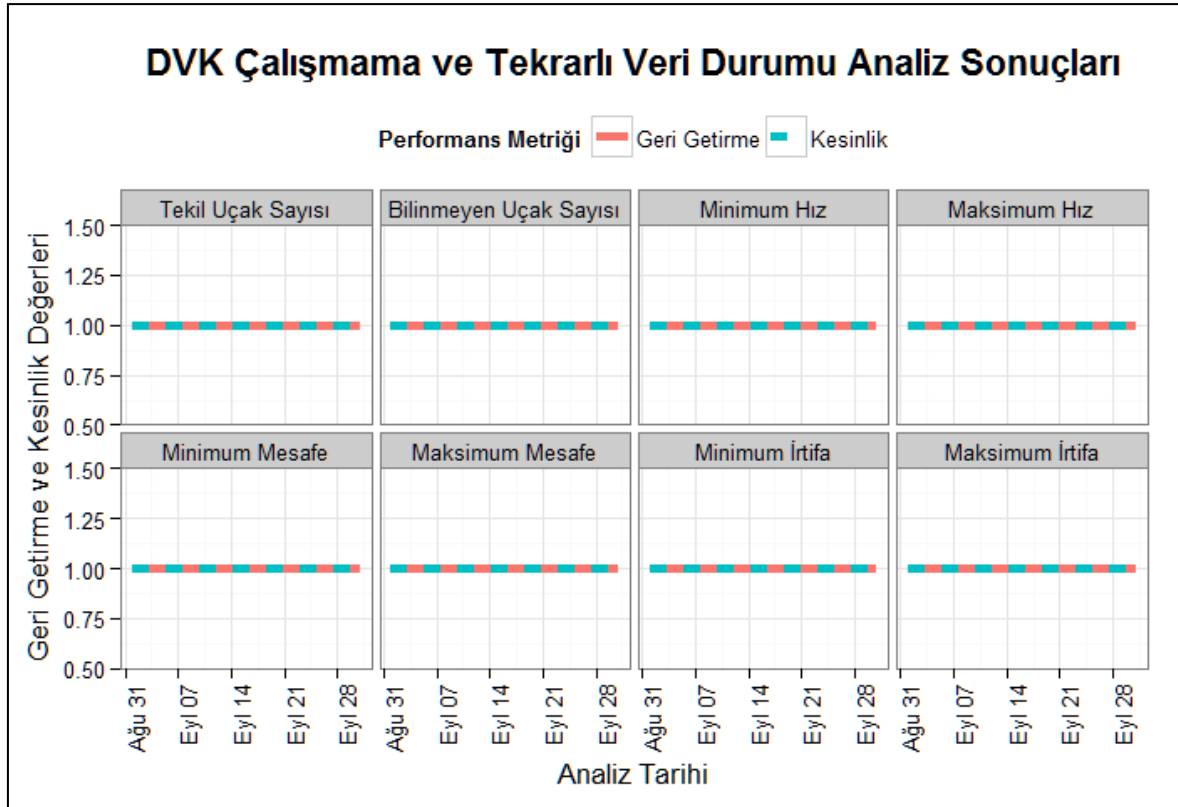
Performans ölçütleri olan kesinlik ve geri getirmenin analiz tarihine (01-30 Eylül 2015) göre değişimi Şekil 4.20’de gösterilmiştir. Analiz bir dakikalık zaman penceresi aralıkları ile gerçekleştirilmiştir. Toplam 43200 zaman penceresinde kesinlik ve geri getirme değerlerinin değişmediği, 1’e eşit olduğu, bu noktadan hareketle zaman pencerelerinin hiçbirinde Lamda Mimarının hata yapmadığı belirlenmiştir.



Şekil 4.19. DVK çalışmadığı ve tekrarlı veri geldiği durumda veri işleme hızının zamana göre değişimi

Daha önceden de bahsedildiği gibi, AVK ve DVK’da verinin işlenmesi aynı yöntem ile yapılmaktadır. Başka bir deyişle bir defa kod geliştirilmekte ve bu kod iki katmanda da aynı şekilde kullanılmaktadır. Verinin birden fazla kaynaktan geliyor olması ile, bir verinin birçok defa tekrarlanması arasında verinin işlenmesi açısından bir fark bulunmamaktadır. Üç farklı kaynaktan verinin zaman farkı ile gelmesi ve bu verilerin tek bir veri haline dönüştürülmesi işleminin geliştirilen algoritma ile yapıldığı daha önce de belirtilmiştir.

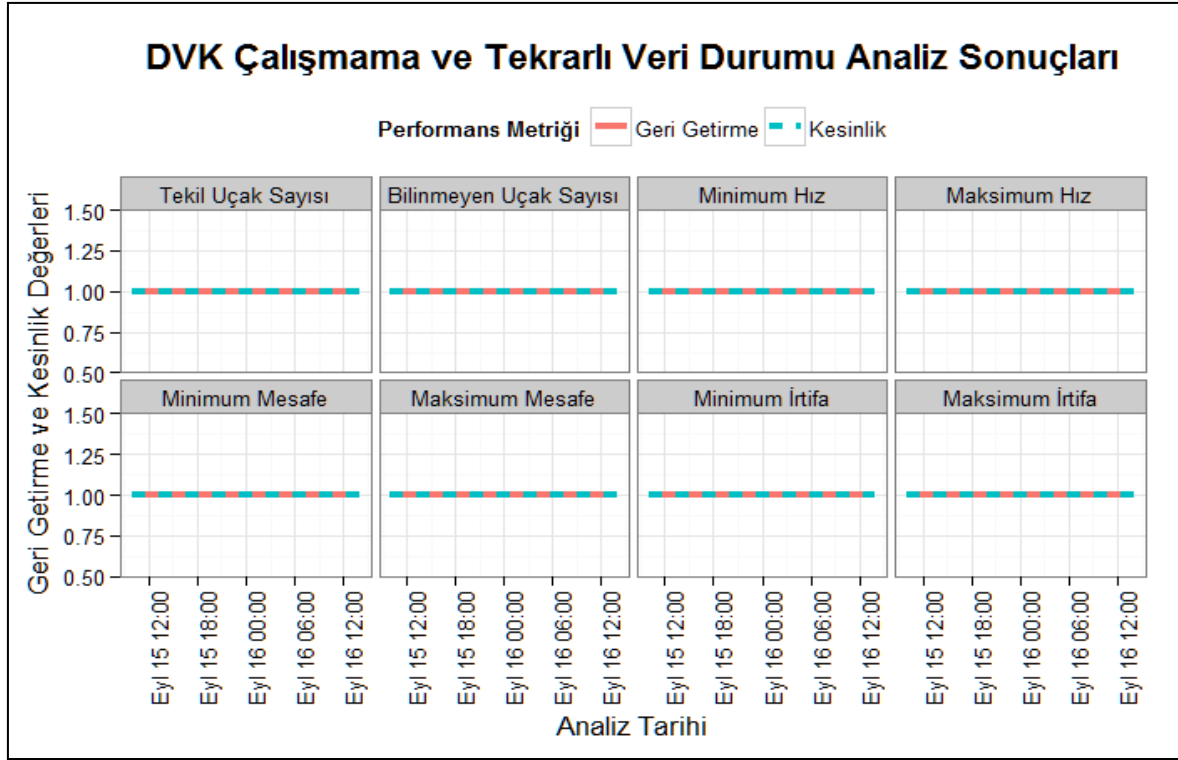
Analiz yöntemi yani algoritma geliştirilirken bu husus dikkate alınarak zaman pencereleri içerisinde indirgeme işlemi uygulanması sayesinde akışkan veri katmanındaki sonuçlarda hata tespit edilmemiştir. Bu hususun dikkate alınmaması durumunda oluşabilecek sonucun, geri getirme değerinin artması ancak kesinlik değerinin de azalması şeklinde olacağı öngörülmektedir.



Şekil 4.20. DVK çalışmadığı ve tekrarlı veri gelmesi durumunda geri getirme ve kesinlik değerlerinin değişimi

AVK tarafından bu şekilde SK'ya iletilen veriler normalden fazla verinin sorgu sonucunda döndürülmesini sağlamayacaktır. Bunun nedeni daha önce tasarım bölümlerinde de anlatıldığı gibi SK'ya verinin belirli bir şema ile ithal edilmesiyle, bu şemada belirtilen işlemlerin yapılıyor olmasıdır. Şemada belirtilen tekillik kestirimi (cardinality estimation) ve minimum-maksimum işlemleri ithal edilen veriye gerçek zamanlı olarak, zaman penceresi tamamlanana kadar uygulanmaktadır. Veri tekrarlı gelse bile buradaki işlemlerden ötürü tekillik kestirimi aynı sonucu verecektir. Veri tekrarlı geldiğinde minimum ve maksimum alma işlemleri sayesinde yine aynı sonuçlar elde edilecektir. Aynı veriden binlerce sayıda geldiğinde bu tekrarlı verilerin minimum ya da maksimumu, bir defa geldiği zamanki minimum ve maksimum ile aynı olacaktır. Dolayısı ile

gerçekleştirilen tasarım sayesinde, SK'da ikinci bir emniyet önlemi olarak şema tabanlı veri ithali kullanılmıştır. AVK katmanında ETL ve diğer işlemlerde bir sorun yaşanmadığı sürece tekrarlı olarak gelen verilerin SK'daki geri getirme ve kesinlik değerlerini etkilemeyeceği öngörülmektedir.



Şekil 4.21. DVK çalışmadığı durumda tekrarlı olarak gelen tarihlerde geri getirme ve kesinlik değerlerinin değişimi

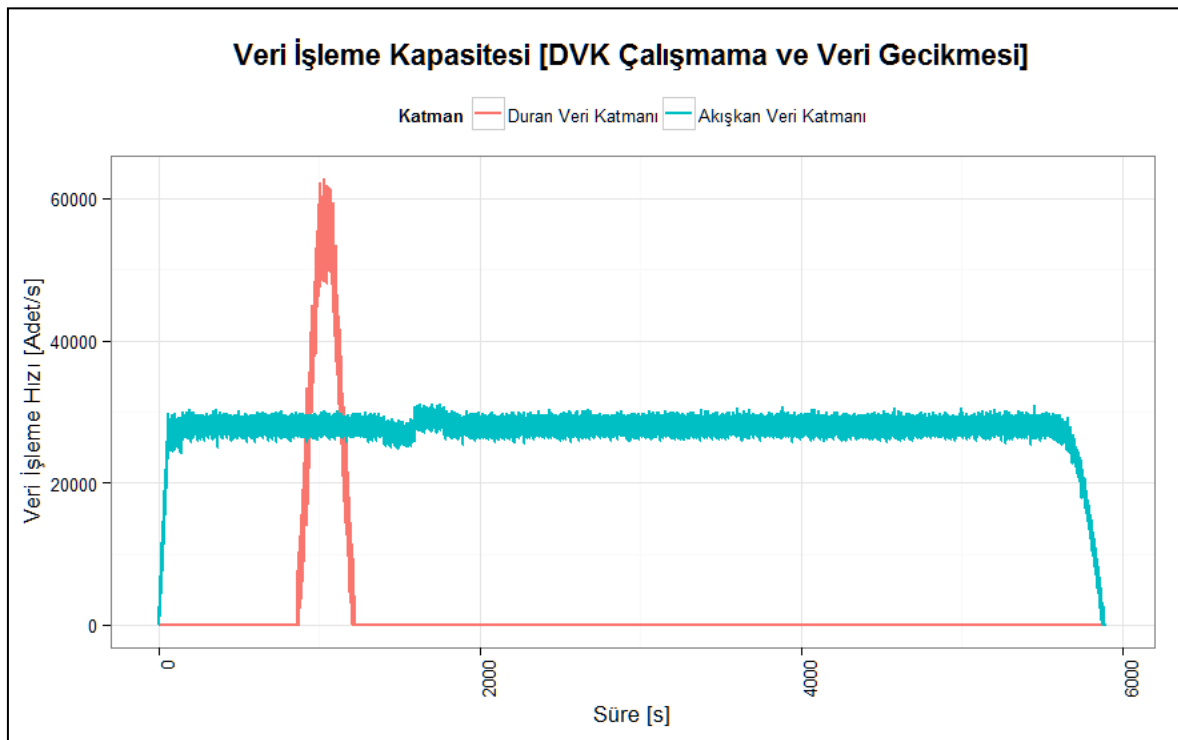
Veri işleme algoritmasının tasarımı esnasında verilerin tekrarlı gelmesi ihtimali önceden dikkate alındığı ve SK'da şema tabanlı veri ithali yöntemi kullanıldığı için SK'daki sorgularda hata oluşmamıştır. Şekil 4.21'de DVK'nın çalışmadığı ve tekrarlı veri gelmesi durumunda, 15 Eylül 2015 saat 10:00 ile 16 Eylül 2015 saat 14:00 tarihleri arasında tekrarlı olarak gönderilen verilerin işlenmesi ile elde edilen analiz sonucu, bu tarih aralığında daha yakından gösterilmektedir.

4.3.7. Duran veri katmanının çalışmaması ve veri gecikmesi durumunda Lamda Mimarinin davranışının incelenmesi

Duran veri katmanının çalışmaması ve veri gecikmesi testinde, Lamda Mimari çalışması gereken nominal durumda incelenmiştir. Veri yolundan alınan veriler sadece akışkan veri

katmanında işlenerek sunum katmanına iletilmektedir. Şekil 4.22’de veri gecikmesi durumunda veri işleme hızının zamana göre değişimi gösterilmiştir. Bu testte, 15 Eylül 2015 saat 10:00 ile 16 Eylül 2015 saat 14:00 tarihleri arasındaki veriler DVK’nın bir defa çalışıp veri işlemeyi tamamlamasından sonraki bir anda geciktirilerek, daha sonra iletilecek şekilde veri yolundan gönderilmektedir.

Lamda Mimarinin tüm veriyi işlemesi yaklaşık olarak 5945 saniye sürdüğü tespit edilmiştir. Bu süre zarfında AVK’nın veri işleme hızının ortalama 27854 adet/s olduğu görülmüştür. DVK’nın bir defalık çalışma süresinin yaklaşık 345 saniye olduğu belirlenmiştir. Bu test gerçekleştirilirken Lamda Mimaride herhangi bir hata oluşmadığı sistem izleme alt yapısı ile tespit edilmiştir. Test beş defa tekrarlanarak sonuçların tutarlılığı kontrol edilmiştir.

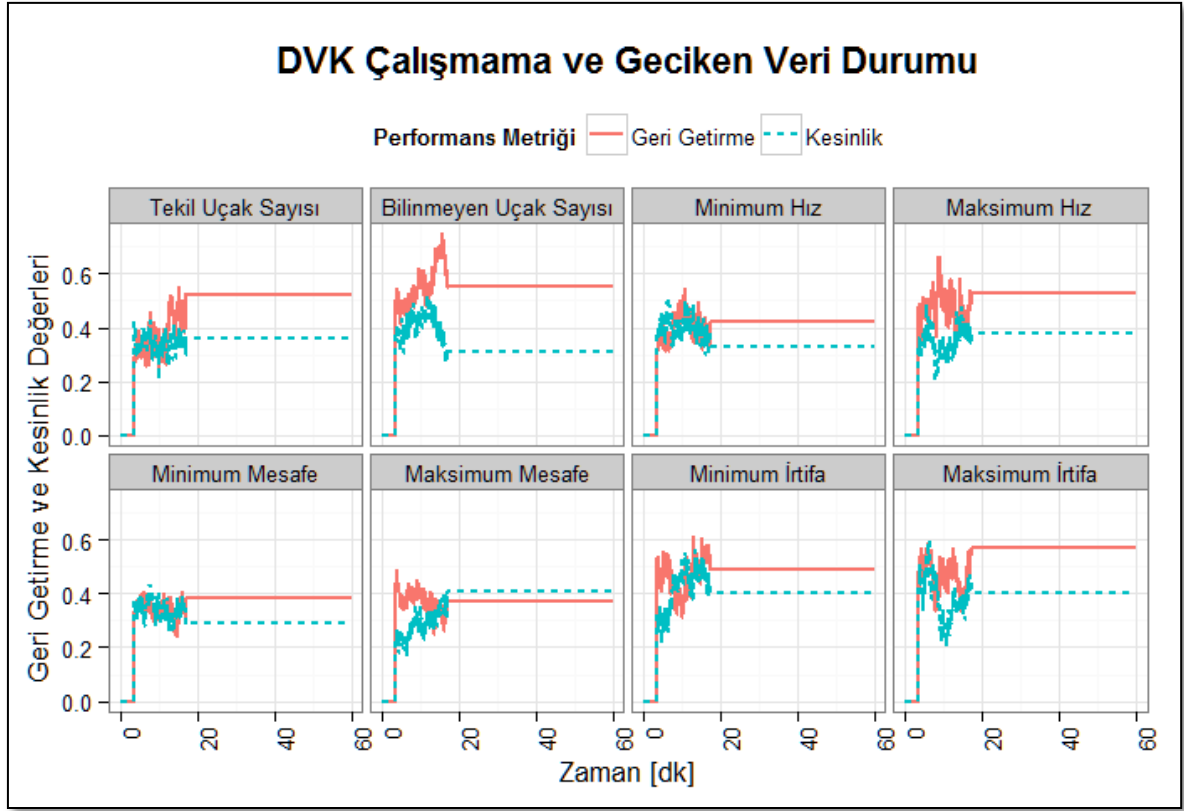


Şekil 4.22. DVK’nın çalışmadığı ve veri gecikmesi durumunda veri işleme hızının zamana göre değişimi

Lamda Mimari olabilecek en kötü durumda test edilmiştir. Şekil 4.23’de DVK’nın çalışmama ve veri gecikmesi durumunda geri getirme ve kesinlik değerlerinin zamana göre değişimi gösterilmektedir. Söz konusu şekilde 15 Eylül 2015 saat 10:00 ile 16 Eylül 2015 saat 14:00 arasındaki kayıtların sorgulanması ile elde edilen ortalama kesinlik ve geri getirme değerlerinin 60 dakika boyunca değişimi gösterilmektedir. Şekilde görüldüğü gibi,

belirli bir süre geri getirme ve kesinlik değerleri sıfır olmakta daha sonra verilerin AVK tarafından işlenmesi ile bu değerler değişmektedir. Bu değişim hiçbir zaman maksimum performans değeri olan 1'e ulaşamamaktadır. Bunun nedeni daha önceden de bahsedildiği gibi SK'daki veri reddetme politikasının geciken verileri işlemeyecek şekilde ayarlanmasıdır. Lamda Mimarinin çalışma yönteminde de belirtildiği üzere, veriler gelmesi gereken zamandan sonra geldiği için işlenerek SK'ya iletilindiğinde SK bu verileri ithal etmemektedir. Yine Lamda Mimarinin çalışma yönteminde belirtildiği üzere, veriler geç gelse bile DVK'da daha sonra işlenmesi için ham haliyle biriktirilmektedir. Ancak DVK'nın bir defa çalışıp daha sonra hiç çalışmamasından ötürü sistem geciken verileri işleyerek SK'daki hatanın giderilmesini asla sağlayamamıştır. 60 dakika boyunca geciktirilen tarih aralığı için yapılan sorgularda verinin işlenmesinden sonra oluşan en son kesinlik ve geri getirme değerleri Lamda Mimarinin çalışmasının sonuna kadar sabit kaldığı gözlemlenmiştir.

Şekil 4.23'de görüldüğü üzere geri getirme ve kesinlik değerleri en çok minimum mesafe ve maksimum mesafe verileri üzerinde hataya neden olmaktadır. Geri getirme ve kesinlik değerlerinin oldukça düşük seviyelerde kaldığı görülmüştür. Daha önceden de belirtildiği gibi Lamda Mimaride işlenmesi en zor metrik minimum ve maksimum mesafedir. Karmaşık hesaplamalar gerektiren bu iki metrik, Lamda Mimaride işlendiğinde AVK'nın daha fazla hata yapmasına, SK'da geri getirme ve kesinlik değerlerinin düşmesine neden olduğu öngörülmektedir. Karmaşık hesaplama işlemlerinin bu gibi durumlarda hataya karşı oldukça hassas olduğu bilinmektedir.



Şekil 4.23. DVK'nın çalışmadığı ve veri gecikmesi durumunda geri getirme ve kesinlik değerlerinin 15 Eylül 2015 10:00 ile 16 Eylül 2015 14:00 zaman aralığı için değişimi

5. SONUÇ

Bu çalışmada Lamda Mimari konseptinin özgün bir şekilde yorumlanarak farklı bir mimari geliştirilmiştir. Geliştirilen yapı literatürdeki çalışmalardan daha kapsamlı biçimde araştırılmış, incelenmiş ve literatürde ilk defa gerçek hayatta kullanılan veriler ile ekstrem durumlar da dahil olmak üzere test edilmiştir. Yapılan araştırmalardan da görüldüğü üzere, henüz olgunlaşmayan ve gerçekleştirilmesi için yeterli teknoloji düzeyi yakalanamamış bir problem olan Lamda Mimarinin konseptine uygun olarak geliştirilmesi özgün özellikler de katılarak başarılmıştır.

Lamda Mimari geliştirilmesi ile ilgili üç ana katman olan duran veri katmanı, hareketli veri katmanı ve sunum katmanının uygun teknolojiler kullanılarak tasarlanması sağlanmıştır. Özellikle sunum katmanında Lamda Mimarinin ihtiyaç duyduğu veri ithal işlemlerini bir arada tam olarak destekleyen bir teknoloji bulunmadığı için, mevcut teknolojiler ile iki farklı veri tabanlarına ithal edilen verilerin sorgulaması ile elde edilen sonuçların bir araya getirilme/birleştirilme zorunluluğu ortadan kaldırılmıştır. Sunum katmanında bu yetenekleri azami olarak destekleyen bir teknoloji kullanılarak, üzerinde yapılan ek geliştirmeler ile tam olarak Lamda Mimari sunum katmanının ihtiyaç duyduğu bir teknoloji haline getirilmiştir.

Lamda Mimarini konseptinin gerekli kıldığı duran ve hareketli veri katmanlarında farklı teknolojilerin kullanılması ve veri işleme için bu teknolojiler kullanılarak iki farklı yazılım/kod geliştirme zorunluluğu ortadan kaldırılmıştır. Bu çalışmadaki teknolojiler ve yöntemler kullanılarak, veri işleme ve algoritma geliştirme için bir defa yazılım/kod geliştirilerek iki farklı katmanda kullanılabilmesinin mümkün olduğu kanıtlanmıştır. Bu yaklaşımla iki farklı katmanın bakım ve idamesinin yapılması ve takip edilmesi yerine sadece bir platformun takip edilmesi gerçek ürün olarak kullanılacak ortamlarda çok büyük faydalar sağlanacağı değerlendirilmektedir. Özellikle bir defa yazılım geliştirmesi ve iki farklı katmanda kullanılması, donanım üzerinde çalışan tek bir kümelemenin kontrol altında tutulması literatürdeki mevcut çalışmalarda şu ana kadar bulunmayan bir özelliktir. Alan spesifik dillerin (DSL) kullanılarak veri işleme üzerine kurgulanmış ve olgun olmayan çok az sayıda çalışma olsa da, bu çalışma mümkün olan en düşük düzeyde doğal platform desteği kullanılarak (native) gerçekleştirilen ilk uygulamadır. Bu nedenle en

doğal geliştirme yöntemi kullanıldığı ve herhangi bir şekilde alan spesifik dillerden doğal dillere dönüştürme işlemleri ve alt yapıları ya da entegrasyon ara birimleri kullanılmadığı için bu yaklaşımın çok daha etkin olduğu belirlenmiştir.

Akışkan veri katmanı üzerinde ayrıca bir gerçek zamanlı uygulama geliştirilerek, akışkan veri katmanındaki gerçek zamanlı verilerin, sunum katmanı kullanılmadan ikinci bir şekilde kullanılabileceği de kanıtlanmıştır. Dolayısı ile Lamda Mimarinin analitik hesaplamalar sonucu elde edilen verilerin sorgulanması özelliğinin yanında gerçek zamanlı veri iletimini de ek bir kaynak tüketmeden yapabildiği gösterilmiştir. Bu özelliği destekleyen literatürde başka bir tasarım bulunmamaktadır.

Gerçek zamanlı olarak yayınlanan ADS-B sinyalleri kullanılarak elde edilen akışkan verilerle yedi farklı alanda, ekstrem durumlar da göz önüne alınarak detaylı testler gerçekleştirilmiştir. Literatürde araştırmasından görüldüğü gibi, söz konusu testlerin mevcut teknolojiler kullanılarak sadece iki tanesi (normal ve azami çalışma durumu) uygulanabilmektedir. Bu çalışma gerçekleştirilen özgün tasarımın olabilecek en kötü durumlar da dâhil olmak üzere yedi farklı senaryoda test edilmesi açısından fark yaratmaktadır. Gerçekleştirilen testlerle, tasarlanan sistemin, Lamda Mimari konseptinin tüm özelliklerini sağladığı kanıtlanmıştır. Ayrıca tasarım testinin yapılabilmesi için Büyük Veriler ile çalışabilen özgün bir test sistemi de geliştirilmiştir. Bu test sistemi yayınlanmış her hangi bir veri setini kullanmamakta ve kendi verisini kendisi temin edebilmektedir.

Detaylı ve titiz yürütülen farklı çalışma koşulu testleri uygulanarak;

- Tasarımın, normal çalışma durumundan çok daha fazla yük altında çalışabildiği, azami veri işleme durumunda bile güvenilir olduğu,
- Normal çalışma durumunda tasarımın güvenilir olduğu ve Lamda Mimari konseptini her katman için tam olarak uyguladığı,
- Normal çalışma durumunda gerçek kullanım senaryolarında oluşabilecek bir durum olan tekrarlı verilerin gelmesi halinde bile sistemin güvenilir çalışabildiği ve tasarımda iki farklı katmanda (AVK ve SK) alınan önlemlerle bu problemin çözüldüğü,
- Normal çalışma durumunda gerçek kullanım senaryolarında oluşabilecek bir durum olan geciken verilerin daha sonra gelmesi halinde, tasarımın Lamda Mimari

konseptine uygun olarak eninde sonunda güvenilir (eventually accurate) çalışabildiği ve DVK'nın oluşan hatayı giderebildiği,

- Duran veri katmanının çalışmaması durumunda tasarımın akışkan veri katmanında bir problem olmadığı sürece güvenilir olarak çalışabildiği, azami yük altında çalışma kapasitesi doğrultusunda tasarım limitleri içinde kalındığı,
- Duran veri katmanının çalışmaması ve tekrarlı veri gelmesi durumunda tasarımın güvenilir olarak çalışabildiği, oluşabilecek bu soruna karşı iki farklı katmanda alınan (AVK, SK) önlemlerin hataya karşı toleransı yükselttiği, başarı sağladığı ve bu koşul altında bile sistemin güvenilir olarak çalıştığı,
- Duran veri katmanının çalışmadığı ve geciken veri gelmesi durumunda, tasarımın, güvenilir olarak çalışmadığı, duran veri katmanı düzeltici görevini yerine getiremediği için tasarımın eninde sonunda kararlı olarak bile çalışmadığı

sonuçları net bir şekilde elde edilmiştir. Çizelge 5.1'de bu konu üzerinde gerçekleştirilmiş çalışmalar ile bu çalışmanın karşılaştırması gösterilmiştir.

Çizelge 5.1. Lamda Mimari akademik çalışma karşılaştırma sonuçları

Karşılaştırma Parametreleri	Bu Çalışma	Demirezen, USMOS [29]	Villary, Alljoyn [27]	Twardowski, Multiagent [32]	Krob, Stream on Demand [34]	Bär, Investigating Lambda [30]	Hasani, Realtime Analytic [33]
Tutarlılık ve hataya karşı dayanıklı olabilme	✓	✓	✓	X	X	✓	X
Düşük gecikmeli okuma ve yazma	✓	✓	✓	?	?	?	X
Ölçeklenebilir olma	✓	✓	X	✓	X	✓	X
Genelleştirilebilme	✓	✓	X	X	X	X	X
Genişletilebilme	✓	✓	X	X	X	✓	X
Anlık sorgulanabilme (ad-hoc)	✓	✓	✓	?	X	?	X
Minimum idame çabası gerektirme	✓	✓	X	X	X	X	X
Hata ayıklanabilme	✓	X	X	X	X	✓	X
Büyük Veri ile test	✓	✓	✓	X	X	✓	X
Büyük Veri ile test ekstrem durumlar	✓	X	X	X	X	X	X
Duran veri katmanı	✓	✓	✓	✓	✓	✓	✓
Akışkan veri katmanı	✓	✓	✓	✓	X	✓	X
Sunum katmanı	✓	✓	✓	X	X	✓	X
Koordinasyon/Kontrol katmanı	✓	✓	X	✓	✓	✓	X
AVK ve DVK için ortak geliştirme	✓	X	X	X	X	X	X
Veri yolu	✓	✓	✓	?	X	✓	X
Duran verinin değişmezliği (immutability)	✓	✓	✓	X	X	✓	X
AVK gerçek zamanlı veri iletim uygulaması	✓	✓	X	X	X	X	X

KAYNAKLAR

1. Marz, N. and Warren, J. (2015). Big Data Principles and best practices of scalable realtime data systems (Birinci Baskı). New York: Manning, 2014.
2. Liu, X., Iftikhar, N., Xie X. (2014). Survey of Real-time Processing Systems for Big Data. ***IDEAS '14 Proceedings of the 18th International Database Engineering & Applications Symposium***, 356-361.
3. Liu, X., Thomsen, C., Pedersen, T. B. (2011). ETLMR: a highly scalable dimensional ETL framework based on mapreduce. ***Proceedings of the 13th international conference on Data warehousing and knowledge discovery***, 96-111.
4. Liu, X., Thomsen, C., Pedersen, T. B. (2012). MapReduce-based dimensional ETL made easy. ***Proceedings of the VLDB Endowment***, 5(12), 1882-1885.
5. Liu, X., Thomsen, C., Pedersen, T. B. (2014). CloudETL: Scalable dimensional for Hive. ***IDEAS '14 Proceedings of the 18th International Database Engineering & Applications Symposium***, 195-206.
6. Dean, J., Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. ***Magazine Communications of the ACM - 50th anniversary issue Communications of the ACM***, 1(51), 107–113.
7. Bange, C., Grosser, T., Anoschek, N. (2013). Big Data Survey Europe: Usage, Technology And Budgets In European Best-Practice Companies. BARC Institute, Wuerzburg, Germany:
8. Lee, K. (2012). Parallel data processing with MapReduce: a survey. processing platforms appeared recently. ***ACM SIGMOD Record***, 40(4), 11-20.
9. Hu, H., Wen, Y., Chua, T.-S. and Li, X. (2014). Towards scalable systems for big data analytics: A technology tutorial. ***IEEE Access***, (2), 652-687.
10. Tatbul, N. (2010). Streaming data integration: Challenges and opportunities, ***Proc. IEEE 26th Int. Conf. Data Eng. Workshops (ICDEW)***, 155-158.
11. İnternet: Apache Storm Project. URL: <http://www.webcitation.org/query?url=http%3A%2F%2Fstorm-project.net%2F&date=2015-12-19>, Son Erişim Tarihi: 19.12.2015.
12. Neumeyer, L., Robbins, B., Nair, A., Kesari, A. (2010). S4: Distributed stream computing platform. ***Proc. IEEE Int. Conf. Data Mining Workshops (ICDMW)***, 170-177.
13. Goodhope, K.; Koshy, J.; Kreps, J.; Narkhede, N.; Park, R.; Rao, J., Ye, V. Y. (2012) (2012). Building LinkedIn's real-time activity data pipeline, ***IEEE Data Eng. Bull***, 35(2), 33-45.

14. Cecchinel, C., Jimenez, M., Mosser, S., Riveill, M. (2014). An Architecture to Support the Collection of Big Data in the Internet of Things., ***Proceedings of the International Workshop on Ubiquitous Mobile cloud, IEEE***, 1-8.
15. Zaharia, M., Tathagata, D., Haoyuan, L., Timothy, H., Shenker, S., Ion, S. (2013). Discretized Streams: Fault-tolerant streaming computation at scale. ***Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles***, 423-438.
16. White, T. (2015). Hadoop: the definitive guide (Dördüncü Baskı). USA: O'Reilly Media Inc., 1-670.
17. Leibiusky, J., Eisbruch, G., Simonassi, D. (2012). Getting started with storm. (Birinci Baskı) USA: O'Reilly Media Inc., 1-250.
18. Nivash, J.P., Deni Raj, E., Dhinesh Babu, L.D., Nirmala, M., Manoj, K.V. (2015). Analysis on Enhancing Storm to Efficiently Process Big Data in Real Time. ***IEEE Computing, Communication and Networking Technologies (ICCCNT)***, 1-5.
19. Kumar, V., Andrade, H., Gedik, B., Wu, K. (2010). DEDUCE: at the intersection of MapReduce and stream processing. ***Proceedings of the 13th International Conference on Extending Database Technology***, 657-662.
20. Demchenko, Y., De Laat, C., Membrey, P., (2014). Defining architecture components of the Big Data Ecosystem. ***2014 International Conference on Collaboration Technologies and Systems (CTS)***, 102-112.
21. Rathore, M.M.U., Paul, A., Ahmad, A., Chen, B.-W. (2015). Real-Time Big Data Analytical Architecture for Remote Sensing Application. ***IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing***, (99), 1-12.
22. Manikandan, S.G., Ravi, S. (2014). Big Data Analysis Using Apache Hadoop. ***2014 International Conference on IT Convergence and Security (ICITCS)***, 1-4.
23. Philip Chen, C.L., Zhang, C.Y., (2014). Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. ***Elsevier Journal of Information Sciences***, (275), 314–347.
24. Kolomvatsos, K., Anagnostopoulos, C., Hadjiefthymiades, S., (in Press). An Efficient Time Optimized Scheme for Progressive Analytics in Big Data, ***Elsevier Journal of Big Data Research***.
25. Pääkkönen, P., Pakkala, D. (in Press). Reference Architecture and Classification of Technologies, Products and Services for Big Data Systems. ***Elsevier Journal of Big Data Research***.
26. Grazzini, J., Pantisano, F. (2015). European Commision, Collaborative research-grade software for crowd-sourced data exploration: from context to practice. Part 1 : Guidelines for scientific evidence provision for policy support based on Big Data and open Technologies, Amsterdam: European Union, 39.

27. Villari, M., Celesti, A., Fazio, M., and Puliafito, A. (2014). AllJoyn Lambda: An architecture for the management of smart environments in IoT. **2014 International Conference on Smart Computing Workshops**, 9-14.
28. İnternet: Kreps, J. Questioning the Lambda Architecture URL: <http://www.webcitation.org/query?url=http%3A%2F%2Fradar.oreilly.com%2F2014%2F07%2Fquestioning-the-lambda-architecture.html&date=2015-12-19>, Son Erişim Tarihi: 19.12.2015.
29. Demirezen, M.U., Küçükayan, Y.G., Acar, D.B., Yılmaz, A.K. (2015). Developing Big Data Realtime Stream And Batch Data Processing Platform. **Processdings of USMOS 2015**, 45.
30. Bär, N. (2014). Investigating the Lambda Architecture. Unpublished Master's Thesis, University of Zurich, Switzerland.
31. Japkowicz, N., Shah, M. (2011). Evaluating Learning Algorithms: A Classification Perspective (Birinci Baskı), UK: Cambridge University Press, 186.
32. Twardowski, B., Ryzko, D., (2014). Multi-agent architecture for realtime big data processing. **IEEE/WIC/ACM International Joint Conferences ob Web Intelligence and Intelligent Agent Technologies(IAT)**, 333-337.
33. Hasani, Z., Velinoc, G. (2014). Lamda Architecture for Realtime Analytic. **ICT Innovstions 2014 Web Proceesings ISSN 1857-7288**, 133-143.
34. Krob, J., Brunnert, A., Prehofer, C., Runkler, A., Krcmar, (2015). Stream processing on demand for lamda architectures. **Springer Computer Performance Engineering**, (9272), 243-257.
35. Sddunçso, M., Calherios, R., Biacnhi, S., Netto, M. (2015). Big Data computing and clouds: trends and future directions. **Elsevier Journal of Distributed Computing**, 79(80), 3-15.
36. Casado, R., Younas, M. (2014). Emerging trends and Technologies in big data processing. **Wiley Concurrency and Computation**, (27), 2078-2091.
37. Ghemawat, S., Gobioff, H., Leung, S-T. (2003). The Google file system. **ACM SIGOPS Operating Systems Review**, 37(5), 29.
38. Lynch, N., Gilbert, S. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. **ACM Special Interest Group on Algorithms and Computation Theory ACM SIGACT**, 33(2), 51-59.
39. Marz, N., Warren, J. (2015). Big Data Principles and best practices of scalable realtime data systems (Birinci Baskı). New York: Manning, 212.
40. Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., Stoica, I. (2010). Spark: cluster computing with working sets. **In Proceedings of the 2nd USENIX conference on Hot topics in cloud computing**, 10-10.

41. Zaharia, M., Das, T., Li, H., Shenker, S., Stoica, I. (2012). Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. ***In Proceedings of the 4th USENIX conference on Hot Topics in Cloud Computing***, 10–10.
42. Casado, R. (2013). Lambdooop, a framework for easy development of Big Data applications. *Conference: NoSQL Matters 2013*, 26.
43. Boykin, O., Ritchie, S., O'Connell, I., Lin, J. (2014). Summingbird: A Framework for Integrating Batch and Online MapReduce Computations. ***Proceedings of the VLDB Endowment***, 7(13), 1441.
44. Goetz, P. T., O'Neill, B. (2014). Storm Blueprints: Patterns for Distributed Real-time Computation (Birinci Baskı), UK: Packt Publishing, 187
45. Goetz, P. T., O'Neill, B. (2014). Storm Blueprints: Patterns for Distributed Real-time Computation (Birinci Baskı), UK: Packt Publishing, 217
46. İnternet: Kimyo, L., Druid Veri İthali ve Tasarım Mimarisi. URL: <http://www.webcitation.org/query?url=http%3A%2F%2Fdruid.io%2Fdocs%2F0.8.1%2Fdesign%2Fdesign.html&date=2015-11-01>, Son Erişim Tarihi: 01.10.2015.
47. Schreuder, J. (2014). Tracking aircraft with PARSAX. Unpublished Master's Thesis, Delft University of Technology, Netherlands.
48. Schafer, M., Strohmeier, M., Lenders, V., Martinovic, I., Wilhelm, M. (2014). Bringing up OpenSky: A large-scale ADS-B sensor network for research. ***IPSN-14 Proceedings of the 13th International Symposium on Information Processing in Sensor Networks***, 83-94.
49. Varga, M., Polgar, Z. A., Hedesiu, H. (2015). ADS-B based real-time air traffic monitoring system.), ***2015 38th International Conference on Telecommunications and Signal Processing (TSP)***, 215-218
50. Boci, E., Thistlethwaite, S. (2015). A novel big data architecture in support of ADS-B data analytic. ***Integrated Communication, Navigation, and Surveillance Conference (ICNS)***, 1-18.
51. Akidau, T., Bradshaw, R., Chambers, C., Chernyak, S. (2015). The Dataflow model: a practical approach to balancing correctness, latency and cost in massive scale unbounded, out of order data processing. ***Proceedings of the VLDB Endowment***, (8)12, 1792-1802.
52. Maier, D., Tufte, K., Papadimos, V., Tucker, P.A. (2005). Semantics and evaluation techniques for window aggregates in data streams, ***ACM Sigmod 2005***, 110-120.
53. Flajolet, P., Fusy, E., Gandouet, O., Meunier, F. (2007). Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. ***Conference on Analysis of Algorithms***, 143.

54. Sahu, P., Dubey, T. A. (2013). Simple scheme for location and distance estimation of wireless sensor nodes in outdoor environments, ***International Journal On Smart Sensing And Intelligent Systems***, 6(1), 368-383.
55. Marz, N., Warren, J. (2015). Big Data Principles and best practices of scalable realtime data systems (Birinci Baskı). New York: Manning, 293.

ÖZGEÇMİŞ

Kişisel Bilgiler

Soyadı, adı : DEMİREZEN, Mustafa Umut
 Uyuğu : T.C.
 Doğum tarihi ve yeri : 10.06.1978, Ankara
 Medeni hali : Bekâr
 Telefon : 0 (312) 418 39 20
 Faks : -
 e-mail : udemirezen@gmail.com



Eğitim

Derece	Eğitim Birimi	Mezuniyet tarihi
Doktora	Gazi Üniversitesi / Fen Bilimleri Enstitüsü, Elektrik – Elektronik Müh. A.B.D.	Devam Ediyor
Yüksek lisans	Gazi Üniversitesi / Fen Bilimleri Enstitüsü, Elektrik – Elektronik Müh. A.B.D.	2003
Lisans	Gazi Üniversitesi / Mühendislik – Mimarlık Fakültesi Elektrik – Elektronik Mühendisliği	2000
Lise	Yüce Fen Lisesi	1995

İş Deneyimi

Yıl	Yer	Görev
2013-Devam	STM A.Ş.	Büyük Veri Grup Lideri
2002-2013	Genelkurmay MEBS Bşk.lığı	Yazılım Mühendisi

Yabancı Dil

İngilizce

Yayınlar

1. Ucuncu, M.; Demirezen, M.U. (2006) Formatted Message Exchange in a Multinational Environment. In Information Fusion for Command Support (pp. 2-1 – 2-10). Meeting Proceedings RTO-MP-IST-055, Paper 2. Neuilly-sur-Seine, France: RTO. Available from: <http://www.rto.nato.int/abstracts.asp>.)
2. M. Selvi, E. B. Aksu, M. H. Dilek, A. Erkan, and M. U. Demirezen, “Attack Tree Visualization for Cyber Security Situational Awareness”, 7th International Conference on Information Security and Cyber Security and Cryptology.
3. ICSC 2015 Conference Tutorial Session “Big Data Analytics and Its Applications for Security and Forensics”.
4. Demirezen M.U, Küçükayan Y.G, Acar D.B, Yılmaz A.K. “Developing Big Data Realtime Stream And Batch Data Processing Platform” Processdings of USMOS 2015.

Hobiler

Piyano, Buz Pateni, Binicilik



GAZİ GELECEKTİR..