T.C.
İSTANBUL ÜNİVERSİTESİ-CERRAHPAŞA
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

Akan Veri İşleyen Dağıtık Sistemlerde Dinamik Ölçekleme

Mert KAVİ

DANIŞMAN
Doç. Dr. Zeynep ORMAN

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

İSTANBUL-2019

Bu çalışma, 27.05.2019 tarihinde aşağıdaki jüri tarafından Bilgisayar Mühendisliği Anabilim Dalı, Bilgisayar Mühendisliği Programında Yüksek Lisans tezi olarak kabul edilmiştir.

Tez Jürisi

Doç. Dr. Zeynep ORMAN(Danışman)
İstanbul Üniversitesi-Cerrahpaşa
Mühendislik Fakültesi

Dr. Öğr. Üyesi Selçuk SEVGEN
İstanbul Üniversitesi-Cerrahpaşa
Mühendislik Fakültesi

Dr. Öğr. Üyesi Akhan AKBULUT
İstanbul Kültür Üniversitesi
Mühendislik Fakültesi

20.04.2016 tarihli Resmi Gazete'de yayımlanan Lisansüstü Eğitim ve Öğretim Yönetmeliğinin 9/2 ve 22/2 maddeleri gereğince; Bu Lisansüstü teze, İstanbul Üniversitesi-Cerrahpaşa'nın abonesi olduğu intihal yazılım programı kullanılarak Lisansüstü Eğitim Enstitüsü'nün belirlemiş olduğu ölçütlere uygun rapor alınmıştır.

ÖNSÖZ

Bu çalışmanın ortaya çıkmasında desteğini esirgemeyen değerli hocam Doç. Dr. Zeynep Orman'a, bugüne kadar akademide ve özel sektörde üzerimde emeği olan tüm hocalarım ve arkadaşlarıma, son olarak hiçbir zaman desteğini esirgemeyen aileme çok teşekkür ederim.

Nisan 2019 Mert KAVİ

İÇİNDEKİLER

Sayfa No

ŞEKİL LİSTESİ

Sayfa No

TABLO LİSTESİ

Sayfa No

SİMGE VE KISALTMA LİSTESİ

Simgeler **Açıklama**

c : İşlemci yükü
C : İşlemci yükü ortalama değeri
H : Hafıza kullanım oranı
M : Hafıza kullanım oranının ortalama değeri
x : Saniyede gelen kayıt sayısı
X : Saniyede gelen kayıt sayısının ortalama değeri
Y : Saniyede gelen kayıt sayısının standart sapma değeri
R : Saniyede gelen kayıt sayısı eşik oranı
Z : Mevcut işleme zamanı
T : Mevcut filigran çıktı zamanı
τ : Gecikme
A : Gecikme süresinin ortalama değeri
Q : Gecikme süresi eşik sabiti
B : Gecikme süresi eşik oranı
J : Lambda değerleri
W : Lambda eşik sabiti
λ : Lambda

Kısaltmalar **Açıklama**

API : Uygulama Programlama Arayüzü

ÖZET

YÜKSEK LİSANS TEZİ

Akan Veri İşleyen Dağıtık Sistemlerde Dinamik Ölçekleme

Mert KAVİ

İstanbul Üniversitesi-Cerrahpaşa

Lisansüstü Eğitim Enstitüsü

Bilgisayar Mühendisliği Anabilim Dalı

Danışman : Doç. Dr. Zeynep ORMAN

Gelişen teknoloji ile birlikte kuruluşlar büyük hacimde ve hızda veri üretmektedir. Sayısı sürekli artan veri kaynakları birçok veri akışı oluşturmaktadır. Oluşan bu verileri gerçek zamanlı olarak işleyebilmek ve analiz edebilmek, uygulamaları devamlı olarak gözlemleyebilmek ve bu sayede müşterilerine kişiselleştirilmiş teklifler vererek ürün önerileri yapabilmek kurumlara rekabet ortamında çok ciddi avantajlar sağlamaktadır. Bununla birlikte, akan veriyi işleyen dağıtık sistemleri inşa etmek ve operasyonunu sağlamak oldukça karmaşık ve maliyetli bir süreçtir. Sistemlerin, veri akışının değişen hızlarına adapte olabilmesi ve gerektiğinde ölçeklenmesi gerekmektedir. Bu nedenle, akan veriyi işleyen dağıtık sistemlere entegre edilecek etkin bir otomatik ölçekleme sistemi kullanılması çoğu zaman kaçınılmazdır. Özellikle, son yıllarda, hızla artan akan veri kaynaklarını işleyebilen sistemlere olan ilgi oldukça artmıştır ve literatürde de bu alanda yapılan çok sayıda çalışma bulunmaktadır. Bu çalışmaların birçoğu sistemin değişen iş yüklerine adapte olabilmesi ve ölçeklenebilirlik konusundan ziyade sistemin olağan şartlarda nasıl çalışacağı üzerine bir takım sistem önerileri üzerine yoğunlaşmıştır. Ölçeklenebilirlik üzerine literatürde az sayıda çalışma bulunmaktadır. Ayrıca, Apache Flink üzerine yapılan çalışma sayısı da oldukça azdır.

Bu tez çalışmasında, yukarıda bahsettiğimiz problemlerden ve literatürdeki bu eksikliklerden yola çıkılarak, Apache Flink üzerinde çalışan değişen çalışma yüklerine adapte olabilen bir sistem tasarımı önerilmiştir. Büyük veri işleyen sistemlere entegre çalışabilecek bu model ile sistem performanslarının geliştirilmesi hedeflenmiştir. Apache Flink hem geliştirme

yapacağımız sistem hem de ölçekleme metriklerini alıp hesaplama yapan bir bileşen olarak kullanılmıştır. Apache Flink metrikleri, Akka kullanılarak Apache Kafka'ya gönderilmiştir. Apache Kafka sistem metriklerini toplayan ve dağıtan bir mesaj kuyruğu olarak konumlandırılmıştır. Apache Flink'in sağladığı metrikler içerisinden en doğru sonuca ulaştıracak metrikler seçilmiş ve oluşturulmuştur. Sistemin hangi durumlarda ölçeklendiği ve ölçeklemeden sonraki durumu analiz edilmiştir. Elde edilen sonuçlar simülasyon çalışmaları ile gösterilerek önerilen sistem metodolojisinin etkinliği test edilmiştir.

Nisan 2019, 60 sayfa.

Anahtar kelimeler: Dağıtık sistemler, büyük veri, akan veri işleme, ölçeklenebilirlik, Apache Flink

SUMMARY

M.Sc. THESIS

Dynamic Scaling At Distributed Data Stream Processing Systems

Mert KAVİ

Istanbul University-Cerrahpasa

Institute of Graduate Studies

Department of Computer Engineering

Supervisor : Assoc. Prof. Dr. Zeynep ORMAN

With the emerging technology, organizations produce data at large volumes and speed. Growing number of data sources create many data streams. The ability to process and analysis these data in real time, monitor the applications continuously and giving personalized offers to customers provide huge competitive advantages to corporations. Establishing large-scale distributed stream processing systems and ensuring their operations is a very complex and costly process. These systems should be capable of adapting the varying rates of data stream and they must be scaled, if required. It is usually inevitable to use an effective automatic scaling system which can be integrated into such systems. In recent literature, there are numerous studies on this issue. Many of these studies have focused on how these systems will operate under normal conditions. There are limited studies on scalability where scaling is usually implemented with a set of resources. In addition, number of studies on Apache Flink is quite few. In this study, based on these shortcomings, a system design which can adapt to changing working loads and work on Apache Flink, is proposed. Apache Flink is used for both system development and calculating the scaling metrics. Scaling is performed by evaluating the expected latency calculated and some critical metrics. It is aimed to improve system performances and reduce quality losses with this model, which can be integrated into big data processing systems. Pre-scaling and post-scaling cases are also demonstrated by simulations to show the effectiveness of the proposed system.

April 2016, 60 pages.

Keywords: Distributed systems, big data, stream processing, scalability, Apache Flink

1. GİRİŞ

Günümüzde kurum ve kuruluşlar çok büyük ölçekte ve hızda veri üretmektedir. Hızla artan veri kaynakları devamlı bir veri akışı oluşturmaktadır. Uygulama sunucularından gelen günlük verileri, web sitelerinden, mobil uygulamalardan gelen tıklama akışları, nesnelerin interneti denen aygıtlardan gelen ölçüm verilerinin tümü, müşterileri, uygulamaları ve ürünleri daha iyi anlamak, tanımak ve analiz etmek için yardımcı olmaktadır. Bu verileri gerçek zamanlı olarak işleyebilmek ve analiz edebilmek, uygulamaları devamlı olarak gözlemleyerek yüksek bir çalışma zamanı sağlamak, kişiselleştirilmiş teklifler vermek ve ürün önerileri yapmak kurumlara çok ciddi avantajlar sağlamaktadır. Akan veriyi işlemek birçok kullanım senaryosunda boy göstermektedir. Web site analitiği ve yapay öğrenme yöntemleri, daha doğru ve aksiyon alınabilir sonuçlar elde etmek için veriyi saatler ve günler yerine dakikalar ya da saniyeler içinde kullanılabilir hale gelmesini sağlayabilmektedir.

Akan veriyi işleyen uygulamalarda genelde iki tür kullanım senaryosu vardır:

- Yığın analitikten akan analitiğe geçiş

Geleneksel veri ambarında veya Hadoop çatılarında verinin yığın halinde işlenmesi yerine gerçek zamanlı analitiğe çevrilmesi, veri gölünün oluşturulması, veri bilimi ve makine öğrenmesi yöntemlerinin kullanılması "yığın analitikten akan analitiğe geçiş" kullanım senaryolarına örnek olarak verilebilir. Akan veriyi veri gölüne atmak, makine öğrenmesi ile oluşturulan modelleri gelen yeni verilerle güncellemek, elde edilen çıktıların doğruluğunu ve güvenliğini arttırmaktadır. Bunun yanısıra, bu yöntemlerin kullanılması güncel veriden faydalanmayı da sağlamaktadır.

- Gerçek zamanlı uygulamalar inşa etmek

Gerçek zamanlı uygulamaların oluşturulması, akan veriyi işleyen servislerin kullanılması ile gerçekleşir. Uygulama gözlemleme, dolandırıcılık tespiti, canlı paneller gibi uygulamalar bunlara örnek olarak verilebilir. Bu kullanım senaryolarını gerçekleyen sistemler, veriyi toplamaktan, işlemeye ve sonuçları sistemlere vermeye kadar olan tüm işlemleri milisaniye

gecikmelerle yapmak zorundadır. Reklam, nesnelerin interneti ve oyun teknolojileri gibi senaryolar bu durumu sağlamak zorundadır.

Veriyi yığın olarak işleme ve akan veriyi işleme arasında bazı temel farklılıklar vardır. Veriyi toplamak, hazırlamak ve işlemek için, veriyi yığın olarak işlemeye göre farklı araçlar kullanılır. Geleneksel analitikte veri alınır ve periyodik bir şekilde veri tabanına yüklenir. Veriyi analiz edebilmek saatler, günler hatta bazen haftalar sürebilir. Akan veriyi işleyen sistemlerde ise veri gerçek zamanlı olarak işlenir, bu işlem verinin saklanmasından önce gerçekleştirilir. Akan verinin hızı ve boyutu sürekli olarak değişir. Akan veriyi işleyen sistemler bu hızın ve varyasyonun güvenli bir şekilde üstesinden gelebilmelidir.

Gerçek zamanlı olarak veriyi işlemek, geleneksel veri işleyen teknolojilere göre çok daha hızlı karar alınmasını sağlar. Fakat bu sistemleri inşa etmek ve operasyonunu sağlamak oldukça karmaşık bir iştir ve sistem kaynaklarının kullanımına sıkı sıkıya bağlıdır. Bu yüzden bu sistemlerin inşası mali açıdan verimli olmak zorundadır. Hesaplama kaynaklarının optimize edilmesi, maksimum verim ve minimum gecikmeyle sistemin çalışması gerekmektedir. Sunucuların yönetilmesi, verinin değişen hızlarına adapte olabilmesi ve ölçeklenmesi gerekmektedir. Tüm bu işlemlerin kötü çalışması zaman ve para kaybını da beraberinde getirmektedir.

Dağıtık veri işleyen sistemlerde karşılaşılan en büyük zorluklardan bir tanesi de sistemin optimize çalışması için gereken kaynakların hesap edilmesidir. Öngörülemeyen veri hacmi ve hız değişimleri, ihtiyaç duyulan kaynak sayısının hesaplanmasını zorlaştırır. Gerekenden fazla kaynak tahsis etmek kaynak israfına yol açarken, düşük sayıda kaynak tahsisi ise sistem performansının düşmesine sebep olmaktadır. Bu nedenle, dağıtık akan veriyi işleyen sistemlere dahil edilecek etkin bir otomatik ölçekleme sistemi kullanılması gerekmektedir.

Bu sistem elastisite prensiplerine uygun olarak servisin çalışma kalitesini ve maliyetleri göz önünde bulundurarak, çalışan sisteme entegre olmalıdır. Elastisite ekonomide bir değişkenin (örneğin; fiyat) başka bir değişkenine göre (örneğin; talep) değişimini ifade eder. Bilgisayarlı hesaplamada ise hesaplama kaynaklarının, dinamik olarak değişen çevresel faktörlere (örneğin; farklı iş yükleri, hesaplama kaynağının maliyeti, kullanıcı istekleri) göre sistemin kalitesini bozmadan üstesinden gelmesi olarak açıklanır.

Akan veriyi işleyebilen birçok dağıtık sistem mevcuttur. Apache Storm, Apache Beam, Apache Spark, Apache Samza ve Apache Flink bunlardan bazılarıdır. Bu tez çalışmasında, yukarıda bahsettiğimiz problemlerden yola çıkılarak Apache Flink üzerinde çalışan değişen çalışma yüklerine adapte olabilen bir sistem önerilmektedir. Bu sistemin en önemli özelliklerinden bir tanesi tüketen tarafın kritik metriklerini kullanarak gecikmeler, işlemci ve hafıza kullanımları, giren ve çıkan veri hacmi gibi birçok veri ölçeklemenin gerçekleştirilmesi için kullanılabilmesidir. Aynı zamanda soyut bir yapı olduğu için birçok dağıtık akan veri işleme sistemi ile kolayca entegre olabilir. Bu özellikler ile önerilen ve geliştirilen sistem hem performans hem de maliyet açısından, dağıtık akan veri işleyen bir sistemi olduğundan daha verimli çalışan bir sistem haline getirmektedir.

Özetle her geçen gün hızla artan akan veri kaynaklarını işleyebilmek büyük önem kazanmıştır. Akan veriyi işleyen sistemler veriyi gerçek zamanlı olarak işler. Gerçek zamanlı olarak işlendiği için sistem çok çeşitli senaryolara adapte olabilmelidir. Kaynakların optimize edilip, çıktının arttırılması temel amaçtır. Bu amaç üzerinde akan veriyi işleyen dağıtık sistemlerde dinamik bir ölçeklemeye gereksinim vardır. Bu tez çalışmasında da yukarıdaki gereksinimler göz önüne alınarak Apache Flink üzerinde dinamik ölçeklemeye olanak sağlayan, entegre olabilir bir harici sistem yapılması hedeflenmiştir. Bu hedef ışığında akan veri işleyen dağıtık bir sistemin metrikleri Java Management Extensions aracılığıyla, Akka kullanılarak Kafka üzerinde toplanmış ve Apache Flink kullanılarak, çeşitli hesaplamalarla sistemin ölçeklenmesi gerektiği durumlar belirlenmiştir.

Bu çalışmanın içeriğinde ilk bölümde, akan veri, dağıtık akan veriyi işleyen sistemler ve ölçeklenebilirlik incelenmiştir. İkinci bölümde, önerilen sistemi gerçekleştirmek için gereken mimari, mimariyi oluşturan bileşenler ve nasıl bir yöntemle ölçeklemenin gerçekleştirileceği, üçüncü bölümde ise örnek bir akan veriyi işleyen dağıtık sistemin simülasyonu yapılarak karşılaştırılmış, sistemin nasıl durumlarda ölçeklendiği ve ölçeklemeden sonraki durumu analiz edilmiştir. Son olarak, elde edilen performans değerlendirmeleri ile elde edilen sonuçlara göre tartışma ve sonuç kısmı yazılmıştır.

2. GENEL KISIMLAR

Dağıtık akan veri işleme sistemleri son yıllarda oldukça popülarite kazanmış ve literatürde üzerine çok sayıda çalışmalar yapılmış bir konudur. Bu çalışmalarda özellikle bulut üzerinde dağıtık sistemleri kullanarak gerçek zamanlı veriyi analiz edebilmek ve sistem ölçekleyebilmek gibi problemler ele alınmıştır. [1-8] çalışmalarında Apache Storm üzerinde operatör tıkanıklıkları ve gecikmeler göz önüne alınarak ele alınan sistemlerin ölçeklenmesi amaçlanmıştır. [1] çalışması sürekli sorguları göz önüne alarak ve aktif operatörlerin yakın gelecekteki metriklerini tahmin ederek, yerel ve global kapsamda paralelizm derecelerini değerlendirir ve değiştirir. Bu çalışmada, klasik Storm ile topoloji gecikmeleri karşılaştırılarak simüle edilmiş ve verimliliği gösterilmiştir. [2] çalışmasında ise çokuzlu ulaşma oranı ya da her işçi düğümü için kaynak kullanımı gibi metriklere bakılarak Storm için varsayılandan farklı bir zamansal programlayıcı geliştirilmiştir. Sistemin verimliliği klasik Storm ile arasındaki servis kalitesinin düşmeden, kaynak kullanımı oranlarına bakılarak gösterilmiştir. [3] çalışmasında kullanıcılara programatik olarak iş akışı tasarlanmasına olanak sağlanmıştır ve sistemin akan verinin içeriğine göre reaktif bir davranış sergilemesi amaçlanmıştır. Çalışmada, sistem konumları göz önüne alınarak belirlenen konumlardaki kaynakların arttırılarak bir kaldıraç etkisi yaratılması yaklaşımını ele almıştır. Bu şekilde gecikmelerin azaltılması sağlanmıştır. Bu yaklaşımın, tek bir konum ile birden fazla konumun karşılaştırılması yapılarak verimlilik sağladığı gösterilmiştir. [4] çalışmasında etkileşim, veritabanı, en son gerçekleştirilen görevleri toplayan bir modül, karar verici algoritmayı sağlayan bir modül, topolojiyi güncelleyen bir modülden oluşan entegre bir çatı tasarlanmıştır. Tasarlanan algoritma kural tabanlı bir yapıda çalışmaktadır. Görev örneklemleri sayıları karşılaştırılarak verimlilik gösterilmiştir. [5] çalışmasında servis kalitesini bozmadan, daha düşük maliyetli çözümler bulunması amaçlanmıştır. Tahmini model temelli bir yaklaşım kullanılarak, sentetik ve gerçek veri kümeleri ile gecikmeler gösterilmiştir. [6] çalışmasında ise, operatör sıralama, yük dengeleme, algoritma seçilimi gibi birçok optimizasyon tekniği ve akan veri işleme örgüsü kullanılarak sentetik yüklerle uygulamalar test edilmiştir. Bu makalede sonuç olarak çalışma örüntüleri üzerinden kaynak kullanım oranları ve maksimum giriş frekansına ve cevap zamanına bakılarak verimlilikler karşılaştırılmıştır. [7] çalışmasında Storm topolojisindeki operatörlerin giriş oranlarına bakılarak bir tıkanıklık metriği hesaplanmıştır. Belirli bir tıkanıklık eşik değerinin üstüne çıkarsa ölçekleme gerçekleştirilmiştir. Önerilen sistem, farklı topoloji tipleri üzerinde denenerek, birim zamanda ortalama çıktı sayısına göre verimlilik gösterilmiştir. [8]

çalışmasında düğümler arasındaki iç trafiği ve işçi düğümler arasındaki etkili olmayan yük dengelerini göz önüne alarak Storm için varsayılandan farklı bir zamanlama algoritması çıkarılması amaçlanmıştır. Çalışma deneysel yüklerle gerçekleştirilmiş ve varsayılan zamanlama algoritması ile karşılaştırılmıştır. Ortalama gecikme ve ortalama düğüm içi trafik karşılaştırılarak verimlilik gösterilmiştir. [9] çalışmasında ise Apache Storm üzerinde statik bir kaynak konfigürasyonu ile topolojileri önceliklendirerek sistem çıktısını arttırmak üzerine çalışılmıştır. [10-12] çalışmalarında Apache Storm üzerinde makine öğrenmesi yaklaşımları kullanılarak kaynakların verimliliğinin arttırılması hedeflenmiştir. [10] çalışmasında dağıtımlar için veri seviyesinde, sorgu planlama seviyesinde, operatör seviyesinde ve küme seviyesinde dört seviye özellik çıkartımı ile birlikte artımlı makine öğrenmesi teknikleri kullanılmıştır. Farklı sorgular, farklı iş yükleri için farklı veri setleri ile model eğitilmiştir. Eğitilen model ile birlikte yeni bir iş yükü geldiğinde ne kadar kaynak tüketileceği makine öğrenmesi modeli ile tahmin edilmiştir. Bu yaklaşım SPS-Storm ile test edilmiştir. Daha düşük işlemci yükü ve hafıza kullanımı gözlenerek verimlilik analiz edilmiştir. [11] çalışmasında olasılık yoğunluğu fonksiyonları kullanılarak, kaynak kullanımı için bir gösterge oluşturması amaçlanmıştır. [10] çalışmasında olduğu gibi burada da geçmişe dayalı sistem kayıtları göz önüne alınarak makine öğrenmesi teknikleri ile model eğitilmiştir. Yapay Sinir Ağlarının özel bir türü olan Karışık Yoğunluk ağları kullanılmıştır. Tahmin edilen ve gerçekte olan işlemci yükü ve hafıza kullanımı oranlarına bakılarak verimlilik gösterilmiştir. [12] çalışmasında ortalama çokuzlu işleme zamanı tahmin edilmek istenmiştir. İş parçacıkları arasındaki gecikme temel ağırlıklardan biri olarak alınmıştır. Eğitimli öğrenme algoriması kullanılarak, işlemci yükü, hafıza, iş parçacığı yükü gibi özellikler kullanılarak sistemin bir öğrenme sürecinden geçmesi sağlanmıştır. Kelime sayma iş yükü kullanılarak topoloji oluşturulmuştur ve ortalama çokuzlu işleme zamanı ile sistemin verimliliği gösterilmiştir. [13] çalışmasında ise, Hidden-Markov modeli kullanılarak değişen yüklere adapte olabilen bir sistem tasarımı önerilmiştir. [14-16] çalışmalarında kullanılan kaynakların kullanımına göre işlerin önceliklendirilmesine dayalı bir sistem tasarlanması amaçlanmıştır. [14] çalışmasında geliştirilen algoritma gözlemleme, analiz, planlama ve yürütme olarak dört fazdan oluşmaktadır. Duruma müdahale etmek için çeşitli hesaplamalar yapılmaktadır ve bu hesaplamalara tahmin yüzükleri adı verilmiştir. Bu yüzükler, analiz ve planlama aşamalarında kullanılmaktadır. Mesaj başına işleme zamanı, mesaj boyutu, bant genişliği kullanımı gibi veriler kullanılmaktadır. Kümenin işlemci ve hafıza kullanımı karşılaştırılarak diğer sistemlere göre önerilen sistemin avantajları gösterilmektedir. [15]

çalışması konuya bir optimizasyon problemi olarak bakmaktadır. Maliyet fonksiyonları ile oluşan bir hesaplama kümesi ile tüm makinelerde servisin kalitesini düşüren olayları azaltmak, işlemci kullanımını kabul edilebilecek bir aralıkta tutmak, aynı kümeyi paylaşan uygulamaların birbirini etkilemesini azaltmak amaçlı bir çalışma yapılmıştır. Çalışma sonucunda önerilen sistem ile birlikte kümenin kaynak kullanımı artmış, ortalama çokuzlu gecikmeler ve servisin kalitesini bozacak olaylar azalmıştır. [17] çalışmasında Apache Storm kaynaklarıyla daha verimli çalışmak için Apache Zookeeper metrikleri göz önüne alınarak alternatif bir önceliklendirme sistem tasarımı geliştirilmiştir. [18, 19] çalışmalarında ise işlenen veri seti üzerinde, verinin anahtarlandığı durumlarda, anahtarlamayı göz önüne alarak popüler anahtarları önceliklendirerek sistemin değişken durumlara adapte olabilmesi amaçlanmıştır. [18] çalışmasında sürece dağıtım aşamasından bakılmıştır. P-Deployer adında bir dağıtım planlayıcı modül oluşturulmuştur. Akan veri işleyen dağıtık sisteme gelen veri hızını dinleyerek, görevler için profiller oluşturulmaya çalışılmıştır. Bu profiller ile birlikte kaynak öngörüsü yapılarak operatörler ölçeklenmeye çalışılmıştır. Kelime sayma iş yükü kullanılarak sentetik bir yük oluşturulmuştur ve sistemin birim zamanda verdiği çıktıya göre verimlilik avantajları incelenmiştir.

Literatürde yapılan bu çalışmalardan farklı olarak bizim önerdiğimiz sistem, akan veri işleyen dağıtık sistemlerden bağımsız, soyut bir şekilde birçok sisteme entegre olabilecek, ölçekleme işlemini Apache Flink üzerinde gerçekleştirebilecek şekilde tasarlanmıştır. Aynı zamanda bağımsız metrik toplama modülü yine kendi içerisinde mevcuttur. Buna ek olarak diğer çalışmalara göre ölçekleme için farklı hesaplama teknikleri ve metrikleri kullanılmıştır.

Tez çalışmamızda özellikle akan veri işleme ve ölçeklenebilirlik konuları önem arz ettiği için öncelikle aşağıda bu kavramların neler olduğundan bahsedilmiştir.

Binlerce veri kaynağından, süreklilik arz edecek bir şekilde, eş zamanlı olarak ve küçük boyutlarda verinin gönderilmesi akan veriyi oluşturur. Akan veri seri bir şekilde veya aşamalı olarak, kayıt bazında veya kayan pencereler şeklinde işlenebilir. Akan veri üzerinde filtreleme, korelasyon, örnekleme gibi birçok analitik işlem gerçekleştirilir. Son zamanlarda finansal analiz, trafik izleme, anomali tespiti, sağlık hizmetleri, sensör verileri ve multimedya alanlarında akan veri işleme uygulamaları artan bir eğri çizmektedir. Akan veri işleme uygulamaları, heterojen, otonom ve global dağıtık veri üreten kaynaklardan veriyi alır, bir araya getirir, analiz eder ve sonuçları gerçek zamanlı olarak üretir. Uygulamalar genellikle Veri-

Görev veya Boruhattı-Paralel olarak tasarlanır. Yapı olarak genellikle yönlü döngüsüz graflardan oluşur. Tüm operatörlerin birleşimi bir graf olarak ifade edilir. Köşeler operatörleri (örneğin kaynak, musluk), uçlar veri, görev ya da boruhattı akışları olarak gösterilir. Durum barındıran operatörler, operatörlerin bir sınıfıdır ve gelen verileri işleyip sistemin iç tarafında saklar. Sistem çıktısı, iç tarafta sağlanan durumun devamlı güncellenmesi ile değişir. Bu operatörlere parçalı-durum ya da anahtarlanmış operatör adı verilir. Giren veri akışı çatallanır ve birden çok alt akışları oluşturur. Bu alt akışlar birbirinden bağımsız olarak işlenir.

Akış işleme üzerine uzun yıllardır aktif olarak araştırma yapılmaktadır. Borealis akış uygulama yazmak için geliştirilen sistemlere birer örnek olarak verilebilir. Borealis Brown üniversitesi ve MIT ile birlikte geliştirilen, akan veri işleyen bir dağıtık bir sistemdir. En son versiyonu 2008 yılında yayınlanmıştır ve artık geliştirilmemektedir. Akış işleyen motor, koordinatör, yük yöneticisi, yük akıtıcısı, hata tolere modülü gibi çok sayıda modüle sahiptir. Yeni nesil geliştirmelerle ile Spark, Storm, Flink gibi daha modern akış işleme motorları ortaya çıkmıştır.

Sistemin, ağın, sürecin artan iş yüklerine veya potansiyel olarak artma ihtimali olan durumların üstesinden gelebilme yeteneğine ölçeklenebilirlik denir. Örneğin bir sisteme artan iş yükü altındayken, yeni bir kaynak ekleyerek toplam çıktı arttırılabiliyorsa bu sistem ölçeklenebilir bir sistemdir. Kelimeyi analojik olarak açıklarsak, ölçeklenebilirlik aslında ekonomi bağlamında kullanılan bir terimdir. Bir şirketin ölçeklenebilirliği, iş modelinin değişmesiyle ekonomik büyümenin aynı potansiyele sahip olmasıdır. Benzer şekilde, dağıtık veri işleyen sistemlerde de ölçeklenebilirlik sistemin zamanla artan veri sayısı ile büyüyebilmesini ifade eder.

Ölçeklenebilirliği genel bir sistem içerisinde tanımlamak genelde çok zordur. Farklı boyutlarda farklı metrikler önem arz eder. Elektronik sistemlerde, veri tabanlarında, ağ alanında çok kritik bir öneme sahiptir. Yeni bir donanım ekleyerek sistemin performansını iyileştirebiliyorsak bu sisteme ölçeklenebilir diyebiliriz. Gerçek hayattan bir örnek verecek olursak, teknolojik bir sistemin 1 ile 1000 kullanıcı arasındaki performansının tüm durumlarda aynı olması o sistemin bileşenlerinin ölçeklenebilir olduğunu gösterir.

Ölçeklenebilirlik farklı boyutlarda, idari ölçeklenebilirlik, fonksiyonel ölçeklenebilirlik, coğrafi ölçeklenebilirlik ve yük bazında ölçeklenebilirlik olarak çok çeşitli şekillerde tanımlanabilir. Farklı ölçeklenebilirlik tanımları aşağıda özetlenmiştir:

- İdari ölçeklenebilirlik: Organizasyon veya kullanıcı sayısı arttıkça tek bir dağıtık sistemle bu isteklere cevap verebilmektir.
- Fonksiyonel ölçeklenebilirlik: Sistemi geliştirmek için eklenecek yeni fonksiyonların minimum eforla eklenebilmesidir.
- Coğrafi ölçeklenebilirlik: Sistemin performansının ve kullanılabilirliğinin yerel alandan daha dağıtık bir coğrafik alana kadar sürdürülebilir olmasıdır.
- Yük bazında ölçeklenebilirlik: Sistemin değişen yüklere kaynak ekleyerek, çıkararak veya düzenleyerek adapte olabilmesidir. *Bu tez konusu için en önemli ölçeklenebilirlik türüdür*.

Bir uygulamaya kaynak eklemeyi ölçekleme olarak düşünürsek, ölçekleme dikey ve yatay ölçekleme olarak ikiye ayrılır. *Yatay ölçekleme,* sisteme bir veya daha fazla düğüm eklemek veya çıkarmaktır. Örnek verecek olursak dağıtık akan veri işleyen sisteme yeni bir bilgisayar eklemek yatay ölçeklemeye bir örnektir. Yüksek işlemci gücü gerektiren işlemler, süper bilgisayarların çok pahalı olması nedeniyle yatay ölçeklenebilir dağıtık sistemlerle ve geleneksel bilgisayarlar ile daha düşük maliyetlerle gerçekleştirilebilmektedir. Yüzlerce küçük bilgisayar bir küme oluşturur ve hesaplama güçlerini birleştirerek beraber hareket edebilir. *Dikey ölçekleme* ise bir düğüme sahip olan sisteme yeni kaynaklar eklemek veya çıkarmaktır. Genelde tek bir bilgisayara ek işlemci veya hafıza eklenmesidir. İki model arasında çeşitli avantajlar ve dezavantajlar vardır. Çok sayıda bilgisayar demek karmaşanın artması demektir. Daha kompleks programlama modelleri, düğümler arasında çıktı ve gecikme problemleri ortaya çıkartır. Bunun yanı sıra bazı uygulamalar bu karmaşıklık ile baş edebilecek dağıtık hesaplama modeline sahip değildir.

3. MALZEME VE YÖNTEM

Dağıtık akan veriyi işleyen sistemler genellikle daha çok Apache Storm, Apache Spark, Apache Kafka, Apache Flink, Akka üzerine gerçeklenmektir. Bu tez çalışmasında Scala programlama diliyle, Apache Flink hem geliştirme yapacağımız sistem hem de ölçekleme metriklerini alıp hesaplama yapan bir bileşen olarak kullanılacaktır. Akka, Apache Flink metriklerini Apache Kafka'ya gönderen olarak konumlanmıştır. Apache Kafka ise sistem metriklerini toplayan ve dağıtan bir mesaj kuyruğu olarak kullanılmıştır. Kullanılan bu araçların yapısı ve işleyişi aşağıda ele alınmıştır.

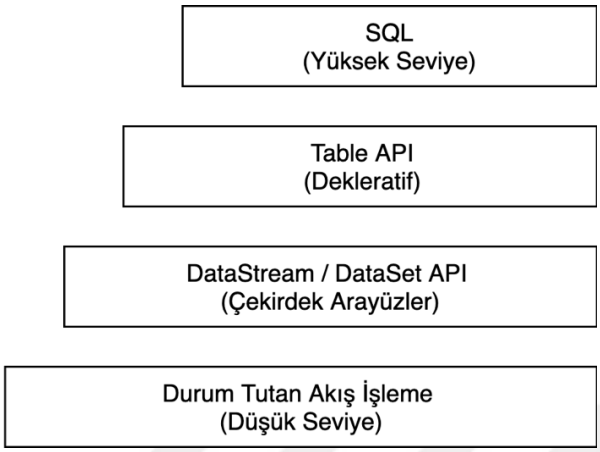
3.1.APACHE FLİNK

Apache Flink açık kaynaklı bir dağıtık akan veri işleme çatısıdır. Akan veriyi işleyen çekirdek dağıtık sistem Java ve Scala ile geliştirilmiştir. Apache Flink yüksek verimlilik, düşük gecikme sağlar ve olay zamanı işleme ve durum yönetimi özelliklerini de beraberinde getirir. Apache Flink uygulamaları hataya tolere edebilir ve kesinlikle bir mesaj ulaştırma semantiğini sağlar.

3.1.1. Soyutlama Seviyeleri

Flink akan ve yığın halde uygulama geliştirmek için gibi farklı seviyelerde soyutlamalar sağlar. Şekil 3.1'de görülen en düşük seviye soyutlama, temel bir durum tutan akış işleme sistemi sunar. *DataStream API* üzerine *Process Function* aracılığı ile kullanıcılara bir veya daha fazla akıştan serbest bir şekilde olayları işlemesine olanak sağlar. Bunun yanında hatayı tolere eden bir durum yönetimi sunar. Olay zamanı ve işleme zamanına göre sofistike hesaplamalar yapılabilir. Pratikte yukarıda bahsedildiği kadar düşük seviye bir programlamaya ihtiyaç duyulmaz. Şekil 3.1'de görülen DataStream API ve DataSet API gibi çekirdek arayüzler kullanılır. Kullanıcıya özel transformasyonlar, join'ler, geliştirmeler, pencereler ve durumlar bu API ile kullanılabilir. Şekil 3.1'deki *Table API* tabloların merkezinde kullanılan bir dekleratif alan özelinde bir dildir. Akışlar arasında dinamik olarak değişen tablolar üzerinden işleme yapılır. *Table API* ilişkisel bir model gibidir. Tablolar ilişkisel veri tabanlarındaki tablolara benzer. *Core API*'ye göre daha az şey ifade edilebilir fakat daha özlüdür yani daha faz kod yazarak aynı noktaya gelinebilir. Flink'in sunduğu en yüksek seviye soyutlama Şekil 3.1'de de görüldüğü gibi SQL'dir. Semantik ve ifade edilebilirlik olarak *Table API*'ye benzer fakat

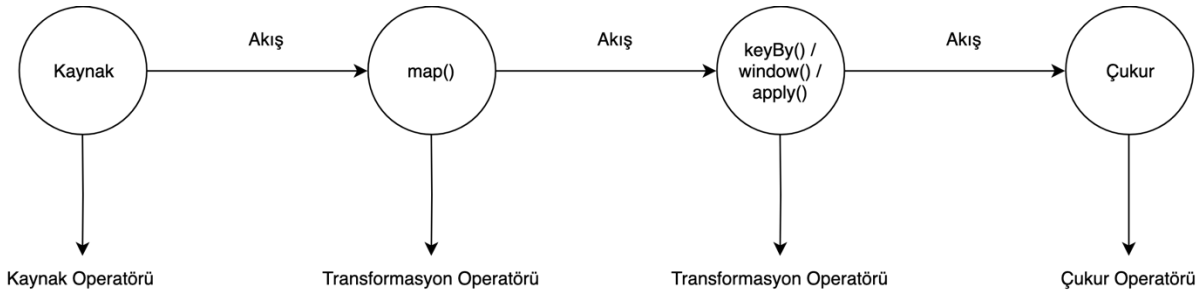
programlar SQL ifadeleri ile sunulur. SQL soyutlama *Table API* ile etkileşim halindedir. SQL ifadeleri *Table API*'nin içerisinde tanımlanan tablolar üzerinde uygulanır.



Şekil 3.1: Apache Flink'de soyutlama seviyeleri.

3.1.2. Programlar ve Veri Akışları

Flink programları akış ve transformasyon bloklarından oluşur. Konsept olarak akış potansiyel olarak asla sonlanmayan veri kayıtlarının akışından oluşur. Transformasyon ise bir veya daha fazla akışı giriş olarak alan, bir veya daha fazla akışı çıkış olarak veren bir operasyondur. Her bir veri akışı Şekil 3.2'de görüldüğü üzere kaynaklar olarak başlar, transformasyonlarla devam eder ve çukurlar olarak biter.



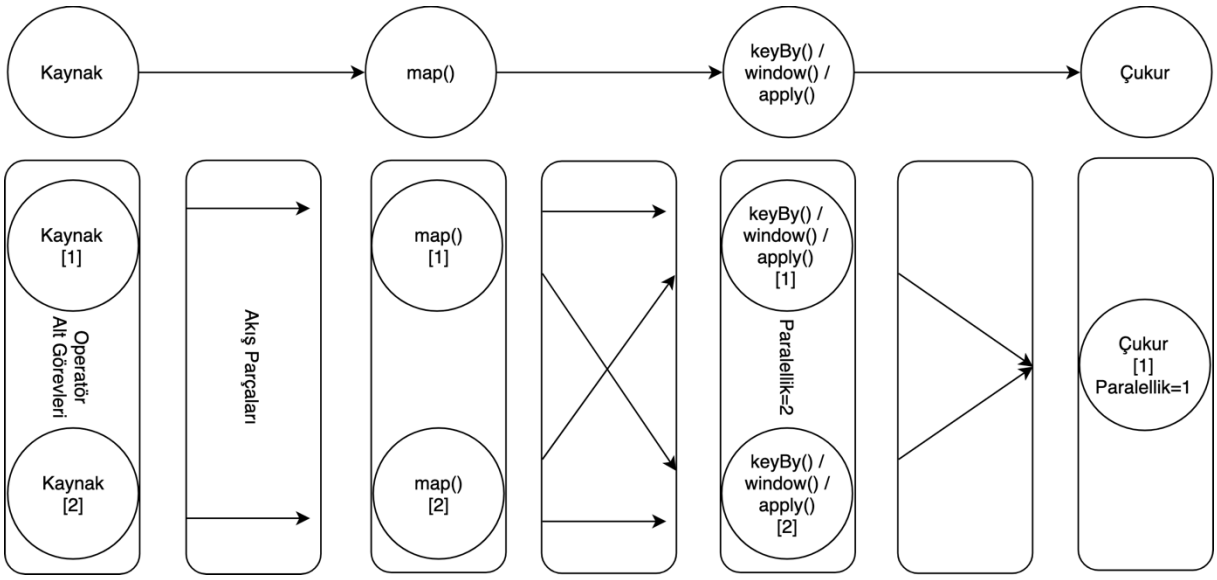
Şekil 3.2: Apache Flink'de veri akışları.

Şekil 3.2'de üç çeşit operatör mevcuttur. Kaynak operatörü, transformasyon operatörü ve çukur operatörü. Kaynak operatörü veri akışına giriş olarak girecek kaynağı belirler. Bu kaynak bir mesaj kuyruğu, bir soket veya bir dosya olabilir. *Map()* transformasyonu bir fonksiyondur. Bir element alır ve bir element üretir. Örneğin *kaynak.map { x => x * 2 }* kod parçası kaynak

içerisindeki tüm elementleri iki ile çarpar ve sonuç olarak kaynak elementlerinin tümü iki ile çarpılmış bir veri üretir. *KeyBy()* transformasyonu bir fonksiyondur. Bir akışı mantıksal olan birden fazla parçalı akışlara dönüştürür. Her parça aynı anahtara sahip elementleri içerir. *KeyBy()* transformasyonu bir *KeyedStream* döndürür. *Window()* transformasyonu bir fonksiyondur. Hali hazır parçalanmış *KeyedStream* üzerine uygulanabilir. Bir anahtarla parçalanmış kaynağı belirli karakteristik durumlara göre (örneğin son beş saniyede gelmiş) pencerelere ayırır. Bir *WindowedStream* döndürür. *Apply()* transformasyonu genel bir fonksiyonu tüm pencereye uygulamayı sağlar. Bir *DataStream* döndürür. Kaynakların giriş olduğu gibi çukurlarda verinin çıkış yerleridir. Üretilen veri akışları dosyalara, soketlere ve mesaj kuyruklarına iletilebilir. Veri akışının çıktılarını sağlamak için geliştirilen bir soyutlamadır.

Flink programları paralel ve dağıtık olarak çalışır. Çalıştırma boyunca bir akış birçok akış parçasından, her bir operatör bir veya daha fazla operatör alt görevlerinden oluşur. Operatör alt görevleri diğerlerinden bağımsızdır ve farklı iş parçacıklarında yürütülür ve muhtemelen farklı makinelerde veya konteynerlerde yer alır.

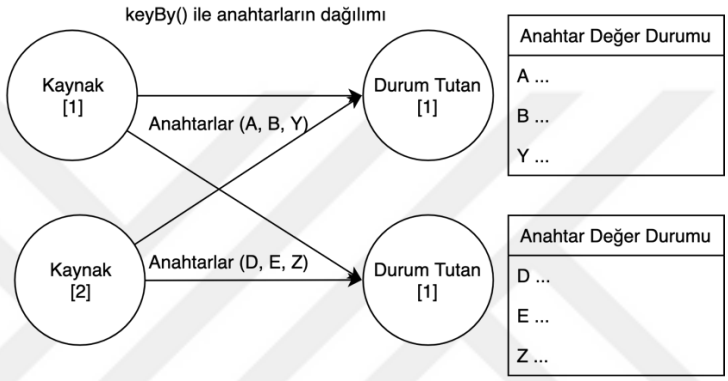
Operatör alt görev sayısı Şekil 3.3'de görüldüğü gibi bir operatörün paralelizm seviyesini belirtir. Aynı programda farklı operatörler farklı seviyede paralelizm seviyelerine sahip olabilir.



Şekil 3.3: Apache Flink'de paralel veri akışları.

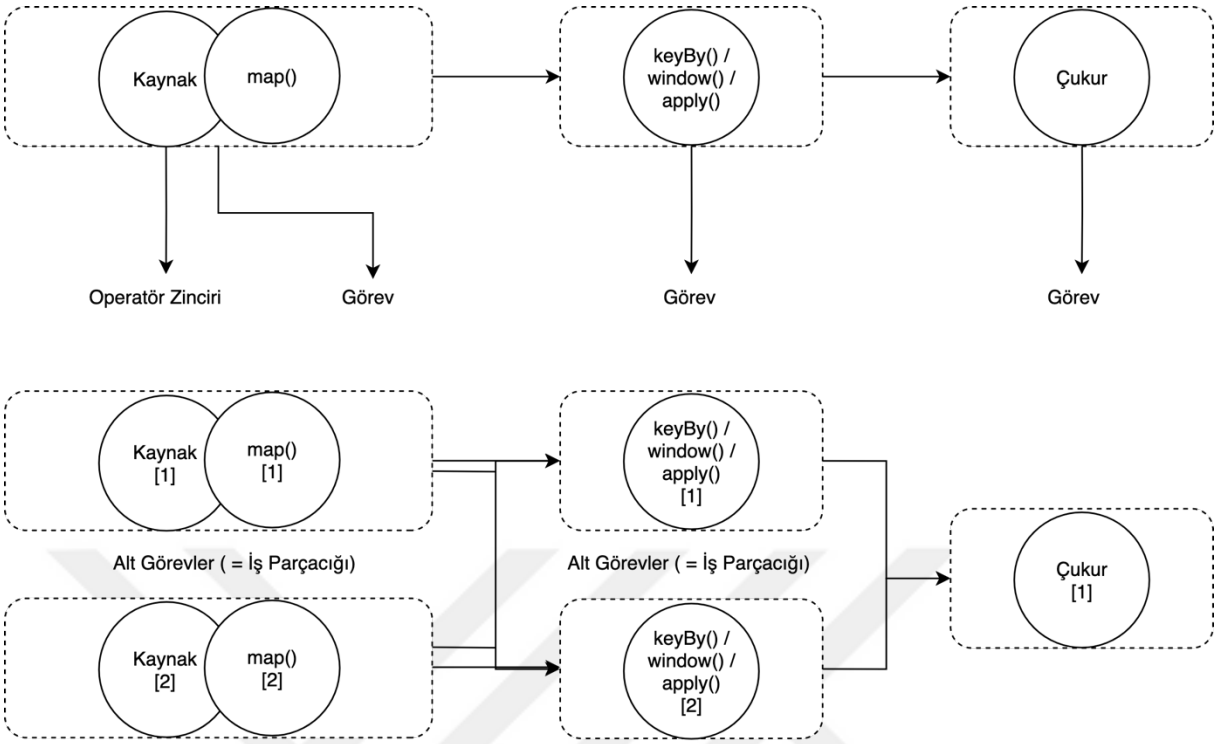
Veri akışındaki çoğu operasyon zaman diliminde yalnızca bireysel olarak bir olay içerir. Bazı operasyonlar ise birden çok olayın bilgisini olaylar arasında hatırlar (örneğin pencere operatörleri). Bu operatörlere durum tutan operatörler adı verilir.

Bu durum tutan operatörler gömülü olarak bir anahtar-değer verisi saklar. Bu veri parçalı ve dağıtık olarak saklanır, durum tutan operatörler tarafından okunur. Anahtar-değer durumuna sadece anahtarlanmış akışlar erişebilir. Bunun için daha önce bahsedilen, Şekil 3.4'de görüldüğü gibi *keyBy()* fonksiyonu kullanılır.



Şekil 3.4: Apache Flink'de durum tutan operatörler.

Flink dağıtık yürütmeyi sağlamak için Şekil 3.5'deki gibi operatör alt görevlerini görevler içerisinde zincirler. Her görev bir iş parçacığı tarafından yürütülür. Operatörleri görevler içerisinde zincirlemek etkili bir optimizasyondur, bir iş parçacığından başka bir iş parçacığına geçmenin yükünü önler, önbelleğe alır ve toplam çıktıyı arttırırken gecikmeyi azaltır. Şekil 3.5'deki örnek veri akışında beş alt görev ile yürütülmektedir ve Flink programının beş iş parçacığı üzerinden yürütüldüğünü de gösterir.



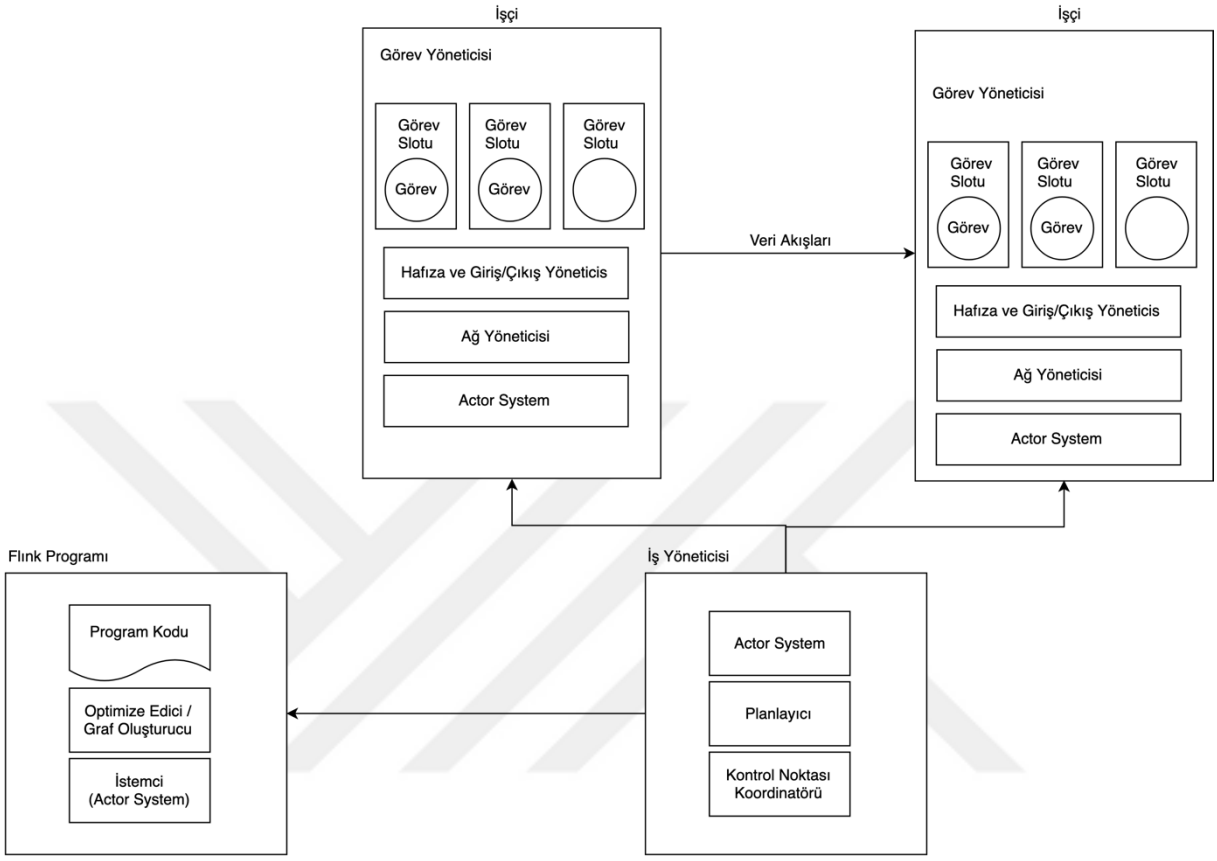
Şekil 3.5: Apache Flink'de örnek bir veri akış diyagramı.

Flink çalışma zamanında Şekil 3.6'daki gibi iki tip süreç yürütür.

- İş yöneticileri dağıtık yürütmeyi koordine eder. Görevleri planlar, kontrol noktalarını koordine eder, hata durumlarından kurtarılmasını sağlar. Her zaman en az bir iş yöneticisi vardır. Yüksek erişilebilir kurulumlarda birden fazla iş yöneticisi bulunabilir. O senaryolarda bir iş yöneticisi lider diğerleri bekleme konumundadır.
- Görev yöneticileri veri akışındaki görevleri yürütür, veri akışlarını önbellekler ve takas eder. Her zaman en az bir görev yöneticisi bulunur.

İş ve görev yöneticileri birçok çeşitli şekilde başlatılabilir. Direkt makineler üzerinde kendi başına başlatılabilir ya da konteynerler içerisinde veya Yarn [20], Kubernetes [21], Apache Mesos [21] gibi kaynak yöneticileri aracılığıyla başlatılabilir. Görev yöneticileri Şekil 3.6'da görüldüğü gibi iş yöneticilerine bağlanır ve erişilebilir olduğunu duyurur. Bunun sonucunda görev yöneticilerine bir iş atanır. İstemci çalışma zamanının ve yürütmenin bir parçası değildir. Görevi veri akışının hazırlanması ve iş yöneticisine gönderilmesidir. Gönderim sonucunda istemci bağlantısını keser veya bağlantıyı sürdürüp raporları alır. İstemci Java veya Scala

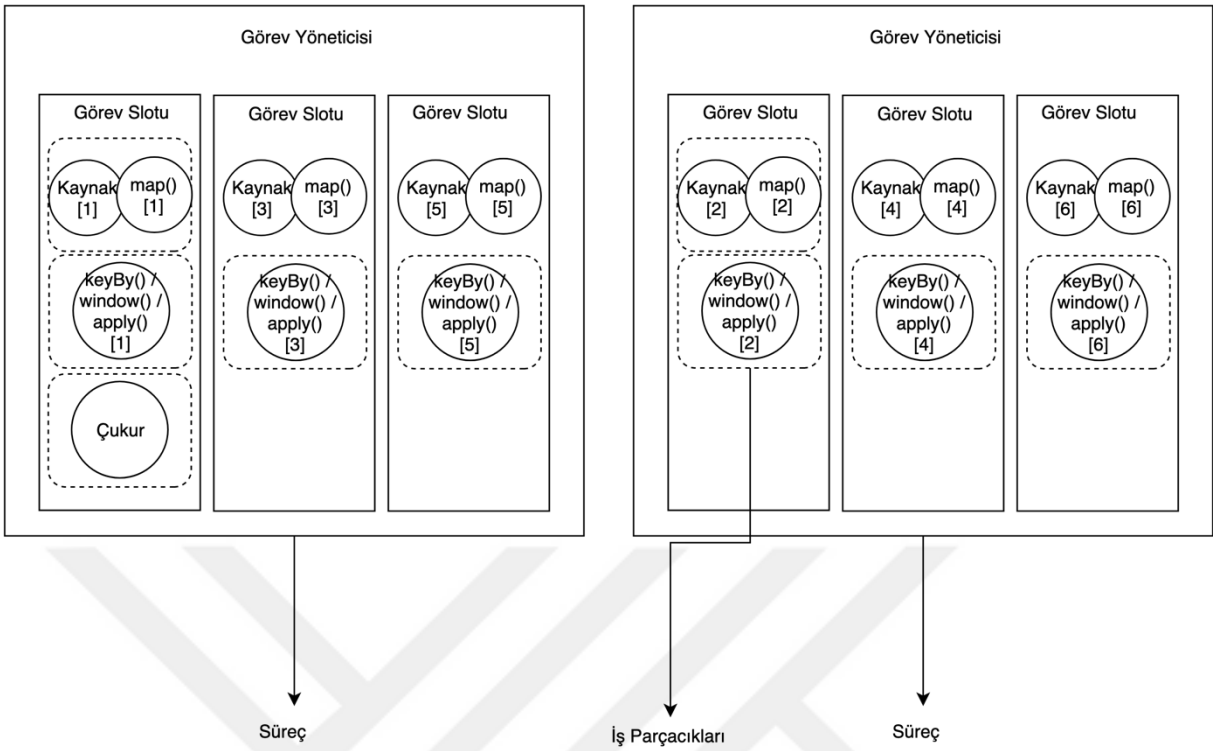
programının bir parçası olarak çalışabilir ya da *./bin/flink run ...* komutu ile komut satırı süreci olarak başlatılabilir.



Şekil 3.6: Apache Flink'de bir programın çalıştırılması.

Her işçi diğer bir adıyla görev yöneticisi bir Java Sanal Makinesi sürecidir ve farklı iş parçacıkkarı üzerinde bir veya daha fazla alt görevi yürütür. İşçiler görev slotlarına sahiptir, bu slotlar aracılığıyla işçinin kaç görev kabul edeceğini belirler.

Her görev slotu Şekil 3.7'de görüldüğü gibi görev yöneticisi kaynağının sabit bir kaynak alt kümesidir. Örneğin bir görev yöneticisi üç adet slota sahip olsun, bu kaynağının hafızasının üçte birine sahip olacağı anlamına gelir. Bir görev yöneticisinde bir slot demek her görev grubunun ayrı bir Java sanal makinesinde koşacağı anlamına gelir. Birden fazla slota sahip olmak ise birden çok alt görevin aynı Java sanal makinesini paylaşması demektir.



Şekil 3.7: Apache Flink'de süreç yönetimi.

3.2.APACHE KAFKA

Apache Kafka dağıtık bir akış platformudur. Üç anahtar özelliğe sahiptir:

- Akışların kayıtlarını yayınlamak ve abone olmak. Bu özelliği ile bir mesaj kuyruğuna veya kurumsal mesajlaşma sistemine benzer.
- Hatayı tolere edebilir ve kalıcı bir şekilde akışların kayıtlarını saklamak.
- Akışların kayıtları oluştuğu anda onları işlemek.

Kafka ile genellikle iki farklı çerçevede uygulamalar gerçekleştirilir:

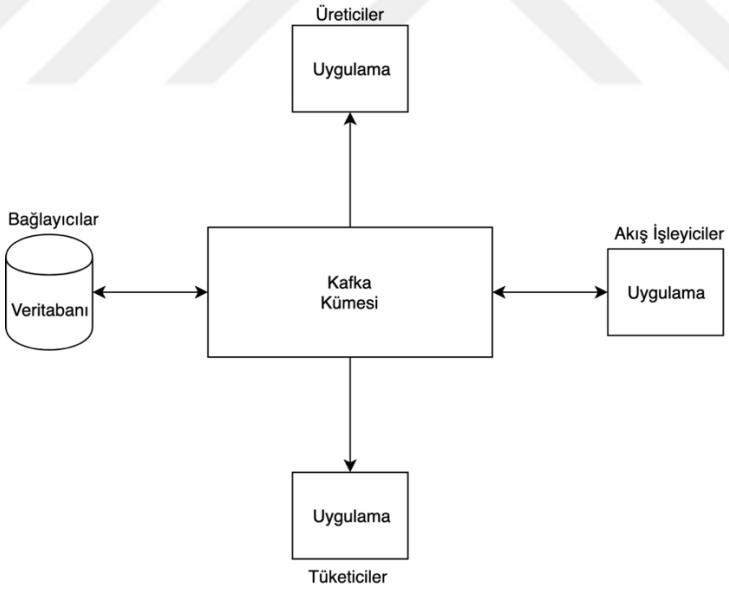
- Güvenli bir şekilde, sistem ve uygulamalar arasında veri alışverişi gerçekleştiren, gerçek zamanlı akan veri boru hatları inşa etmek.
- Akışların verilerini gerçek zamanları olarak dönüştüren ve harekete geçiren uygulamalar inşa etmek.

Kafka bu uygulamaları sağlarken, temel birkaç konsepti kullanır:

- Kafka bir veya daha fazla sunucu ile kümeler şeklinde çalışır.
- Kafka akış kayıtlarını kategoriler altında tutar ve bu kategorilere *topic* adı verilir.
- Her kayıt bir anahtar, bir değer ve bir zamanpulundan oluşur.

Kafka Şekil 3.8 gibi dört çekirdek API'ye sahiptir:

- Üretici (Producer API): Bir veya daha fazla Kafka topic'ine kayıt göndermek için kullanılır.
- Tüketici (Consumer API): Uygulamanın bir veya daha fazla topic'e abone olmasını ve akış kayıtlarını işlemesini sağlar.
- Akış (Streams API): Uygulamanın bir akış işleme sistem olarak davranmasına olanak tanır.
- Bağlayıcı (Connect API): Güvenli ve tekrar kullanılabilir üretici ve tüketiciler oluşturulmasına olanak sağlar. Örneğin ilişkisel veritabanına bağlanan bir uygulama tablodaki her değişimi yakalayıp bir Kafka topic'ine kayıtları gönderir.



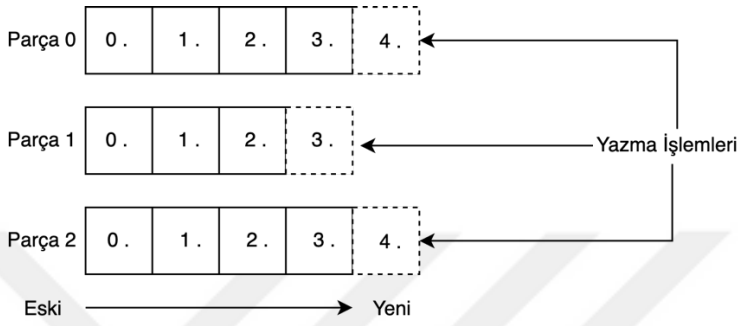
Şekil 3.8: Kafka kümesi ve bileşenleri arasındaki ilişki.

Kafka'da istemciler ve sunucular arasındaki iletişim basit, yüksek performanslı ve dil bağımsız TCP protokolü ile gerçekleştirilir.

3.2.1. Konular ve Günlükler

Konuların her zaman çok fazla abonesi olabilir. Konular sıfır, bir veya daha fazla tüketiciye sahip olabilir.

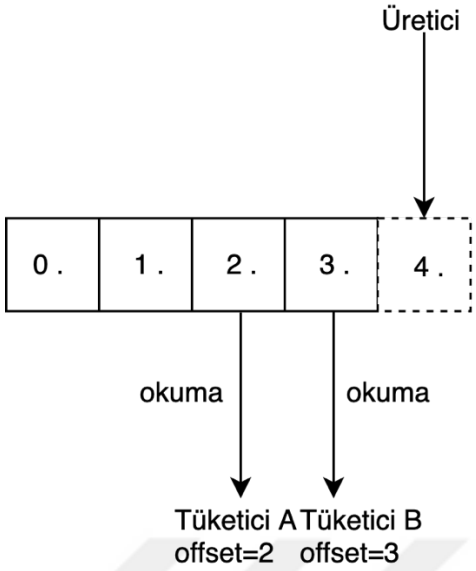
Kafka her bir konu için Şekil 3.9'daki gibi parçalı bir günlük tutar.



Şekil 3.9: Kafka üzerinde konuların anatomisi.

Her bir parça sıralı, değişmez bir kayıt serisinden oluşur ve Şekil 3.10'daki gibi sürekli sonuna eklenen yapısal bir günlüktür. Parçalarda her bir kayıt için bir sıra numarası atanır bu numaralara ofset adı verilir. Günlük parçasındaki her bir kaydın yerini belirtir ve özgündür.

Kafka kümesi yayınlanan tüm kayıtları tüketilsin veya tüketilmesin, ayarlanabilir belirli bir muhafaza süresince kalıcı olarak saklar. Örneğin muhafaza süresi iki gün olarak belirlenmişse, yayınlanan kayıt iki gün boyunca tüketilebilir olacaktır fakat iki gün sonra kayıt diskten silinecektir.



Şekil 3.10: Kafka'da üretici ve tüketicinin çalışması.

Kalıcı olan tek meta verisi, tüketicilerin günlükte kaldığı ofseti ya da pozisyonudur. Tüketici bulunduğu ofset değerini sıfırlayabilir veya daha eski bir değere alabilir. Böylece geçmiş veriyi tekrar işleyebilir.

Kafka günlük parçalarının amacı günlük bloğunun mevcut sunucuya sığması içindir. Bu aynı zamanda paralelizm seviyesini gösterir. Böylece Kafka kümesi daha ölçeklenebilir hale gelir.

3.3. SCALA

Scala fonksiyonel programlama ve güçlü bir statik tip sistemi desteğiyle gelen genel amaçlı bir programlama dilidir. Daha özlü ve Java'nın tartışılan taraflarını geliştirmek için tasarlanmıştır.

Scala kaynak kodu Java bayt koda derlenir, bunun sonucu olarak yürütülebilir kod Java Sanal Makinesi üzerinde çalışır. Scala kodu içerisinde Java kodu çalıştırılabilir ve Java ile yazılan kütüphaneler kullanılabilir. Scala fonksiyonel programlamanın birçok özelliğine sahiptir, Scheme, Haskell gibi dillerden, *currying, type inference, immutability, lazy evaluation ve pattern matching* gibi özellikleri almıştır. Cebir veri tipleri, kovaryans ve kontravaryans, yüksek seviye tipler ve bilinmeyen tipler gibi gelişmiş bir tip sistemine sahiptir. Dilin tasarımı 2011 yılında Martin Odersky tarafından başlamıştır.

3.4.AKKA

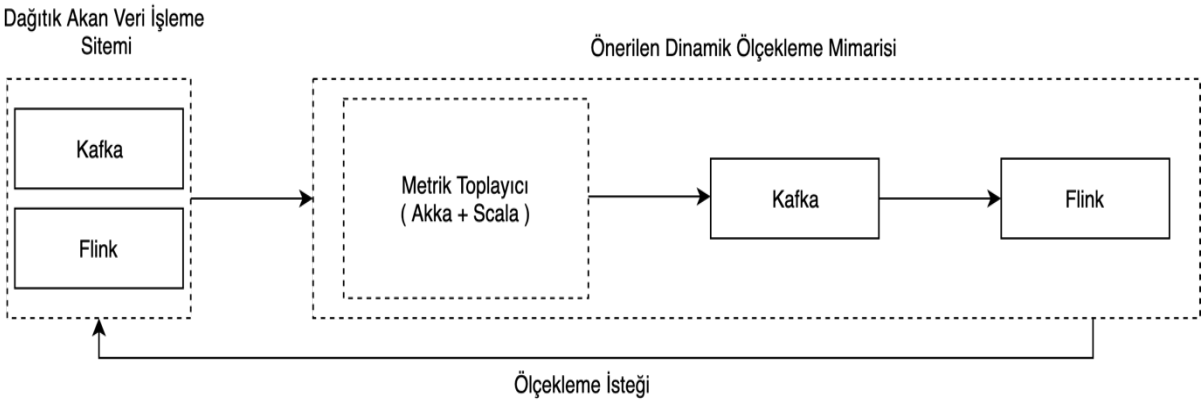
Akka ölçeklenebilir, elastik sistemler tasarlamak için geliştirilmiş açık kaynak kütüphaneler bütünüdür. Akka düşük seviye kod yerine işi gerçekleştirmek için gereken mantığa odaklanmaya olanak sağlar. Bununla beraber güvenli, hatayı tolere edebilen ve yüksek performanslı bir ortamı beraberinde getirir.

Dağıtık sistemlerde başarılı olabilmek için, bileşenlerin cevap vermeden kırılmaması, ağ üzerinde bir mesaj kaybının olmaması, ağ dalgalanma sorunlarının önüne geçilmesi gerekir. Bu durumlara Akka aşağıdakileri sağlar:

- Atomikler ve kilitleri gibi düşük seviye eş zamanlılık yapılarını kullanmadan çok iş parçacığı ile çalışabilmek.
- Sistemler ve onların bileşenleri arasında uzaktan ve transparan iletişim kurmak için zorlayıcı bir ağ kodu yazmamak ve bakımını yapmamak.
- Kümelenmiş, yüksek erişilebilir bir mimariye daha kolay sahip olmak, gerçek bir reaktif sisteme sahip olmak.

Eş zamanlı, paralel ve dağıtık bir uygulama yazabilmek için Akka *actor model* kullanır. Bu model tüm Akka kütüphanelerinin merkezindedir.

3.5. ÖNERİLEN SİSTEM TASARIMI



Şekil 3.11: Çalışma kapsamında önerilen mimari.

Şekil 3.11'de birinci kısım, yani Dağıtık Akan Veri İşleme Mimarisi ölçekleyeceğimiz sistemi

göstermektedir. Burada işlenecek mesajların bulunduğu bir Kafka kümesi ve bu mesajları işleyen bir Apache Flink kümesi bulunmaktadır. Şeklin geriye kalan kısım tamamen dağıtık akan veri işleyen sistemlerin ölçeklenmesi için önerilen sistemin mimarisine aittir. Apache Flink kümesinin metriklerini belirli aralıkları alan ve Kafka kümesine atan bir metrik toplayıcı modülü mevcuttur. Ölçekleme hesabını yapacak verilerin tamamı buradan oluşur. Alınan bu metrikler Kafka kümesine gönderildikten sonra bir Apache Flink kümesi bu metrikleri gerçek zamanlı olarak dinler ve veri parçaları üzerinde çeşitli hesaplamalar yapar. Bu hesaplamaların sonucunda veriyi işleyen Apache Flink kümesine REST API ve HTTP protokolü aracılığıyla ölçekleme isteği gönderir. Sistemin ölçeklenmesini tetikleyecek temel kaynak Apache Flink metrikleri ve kritik hesaplamalardır.

Apache Flink metrikleri toplama ve harici sistemlere almak için bir metrik sistemi sunar. Herhangi bir *RichFunction'dan* türeyen herhangi bir kullanıcı fonksiyonu ile *getRuntimeContext().getMetricGroup()* çağırılarak metrik sistemine istenilen bir yerden erişilebilir. Bu metod bir *MetricGroup* nesnesi döndürür. Bu nesne aracılığıyla yeni bir metrik yaratıp, kaydedilebilir. Flink sayaç, ölçü, histogram, metre tipinde metrikleri destekler.

Sayaç bir şeyleri saymak için kullanılabilir. Mevcut değer *inc()* aracılığıyla arttırılabilir veya *dec()* aracılığıyla azaltılabilir.

Ölçü tipi ile herhangi bir anda herhangi bir tipin değeri talep edilebilir. Bu metrik tipini kullanabilmek için öncelikle *org.apache.flink.metrics.Gauge* arayüzünün implemente edilmesi gerekir. Dönecek değer ile ilgili herhangi bir tip kısıtlaması yoktur.

Histogram metriği *long* tipinde bir değerin dağılımını verir. *MetricGroup* üzerinde *histogram(String name, Histogram histogram)* metodu aracılığıyla kullanılır.

Metre tipinde metrikler, ortalama miktarı ölçümler. Bir olay gerçekleştiğinde *markEvent()* metodu aracılığı ile kaydedilir. *MetricGroup* içerisinde *meter(String name, Meter meter)* metodu ile kaydedilir.

Flink üzerinde sunulan metrik değişkenlerinin listesi aşağıdaki gibidir:

- **JobManager:** Host ile beraber kullanılır.
- **TaskManager:** Host ve taskmanager kimliği ile beraber kullanılır.

- **Job:** Job kimliği ve job ismi ile beraber kullanılır.
- **Task:** Task kimliği, task ismi, task girişim kimliği, task girişim sayısı, sub task ile beraber kullanılır.
- **Operator:** Operator kimliği, operator ismi, sub task ile beraber kullanılır.

Flink metrikler birçok raporlama arayüzünü destekler. Biz bu çalışmada Java Management Extensions raporlayıcısını kullanacağız. Bu raporlayıcı uygulamadan bağımsız, sistem yöneticilerine veya geliştiricilere yardımcı olmak amacıyla ortaya çıkmıştır. Kapasiteyi yönetmek ve problemleri tespit etmek için kullanılır. Flink'de olduğu gibi özelleştirilmiş raporlar da oluşturulabilir. Flink varsayılan olarak, var olan durumu derinlemesine analiz yapmak için birçok önemli sistem metriğini kendisi sağlar.

Tablo 3.1 - Tablo 3.11 tablolarında aşağıdaki kolonlar mevcuttur:

- **Alan:** JobManager, taskmanager gibi alanların hangilerini desteklediğini gösterir.
- **İçtakı:** Alan içerisinde hangi takı ile gösterileceğini gösterir.
- **Metrik:** Hangi metrikleri sağladığını gösterir.

Tablo 3.1: İşlemci metrikleri.

Alan	**Metrik**
Job-/TaskManager	Load
	Time

Tablo 3.1'de Status.JVM.CPU iç takısı kullanılarak Java Sanal Makinesi üzerindeki işlemci kullanımı metrikleri gösterilmektedir.

Tablo 3.2: Hafıza metrikleri.

Alan	**Metrik**
Job-/TaskManager	Heap.Used
	Heap.Committed
	Heap.Max
	NonHeap.Used
	NonHeap.Committed
	NonHeap.Max
	Direct.Count
	Direct.MemoryUsed
	Direct.TotalCapacity

	Mapped.Count
	Mapped.MemoryUsed
	Mapped.TotalCapacity

Tablo 3.2'de Status.JVM.Memory iç takısı kullanılarak Java Sanal Makinesi üzerindeki hafıza kullanımı metrikleri gösterilmektedir.

Tablo 3.3: İş parçacığı metrikleri.

Alan	**Metrik**
Job-/TaskManager	Count

Tablo 3.3'de Status.JVM.Threads iç takısı kullanılarak Java Sanal Makinesi üzerinde iş parçacığı sayısı metriklerini gösterilmektedir.

Tablo 3.4: Çöp toplayıcısı metrikleri.

Alan	**Metrik**
Job-/TaskManager	<GarbageCollector>.Count
	<GarbageCollector>.Time

Tablo 3.4 Status.JVM.GarbageCollector iç takısı kullanılarak Java Sanal Makinesi üzerinde çöp toplayıcısı metriklerini göstermektedir.

Tablo 3.5: Sınıf yükleyicisi metrikleri.

Alan	**İçtakı**	**Metrik**
Job-/TaskManager	Status.JVM.ClassLoader	ClassesLoaded
		ClassesUnloaded

Tablo 3.5 Status.JVM.ClassLoader iç takısı kullanılarak Java Sanal Makinesi üzerinde sınıf yükleme metriklerini göstermektedir.

Tablo 3.6: Ağ metrikleri.

Alan	**Metrik**
TaskManager	AvailableMemorySegments
	TotalMemorySegments
Task	inputQueueLength
	outputQueueLength
	inPoolUsage
	outPoolUsage
	totalQueueLen
	minQueueLen
	maxQueueLen
	avgQueueLen

Tablo 3.6 Status.Network, Buffers, Network.<Input|Output>.<gate> iç takısı kullanılarak Java Sanal Makinesi üzerinde ağ metriklerini göstermektedir.

Tablo 3.7: Küme metrikleri.

Alan	**Metrik**
JobManager	numRegisteredTaskManagers
	numRunningJobs
	taskSlotsAvailable
	taskSlotsTotal

Tablo 3.7 iş yöneticisine ait sisteme kayıtlı görev yöneticisi sayısı, koşan iş sayısı gibi metrikleri göstermektedir.

Tablo 3.8: Erişilebilirlik metrikleri.

Alan	**Metrik**
Job	restartingTime
	Uptime
	downtime
	fullRestarts

Tablo 3.8 işe ait çöküş zamanı ve işin yeniden başlama sayısı metriklerini göstermektedir.

Tablo 3.9: Kontrol noktalama metrikleri.

Alan	Metrik
Job	lastCheckpointDuration
	lastCheckpointSize
	lastCheckpointExternalPath
	lastCheckpointRestoreTimestamp
	lastCheckpointAlignmentBuffered
	numberOfInProgressCheckpoints
	numberOfCompletedCheckpoints
	numberOfFailedCheckpoints
	totalNumberOfCheckpoints
Task	checkpointAlignmentTime

Tablo 3.9 işe ve göreve ait kontrol noktası metriklerini göstermektedir.

Tablo 3.10: Giriş/çıkış metrikleri.

Alan	Metrik
Job	<source_id>.<source_subtask_index>.<operator_id>.<operator_subtask_index>.latency
Task	numBytesInLocal
	numBytesInLocalPerSecond
	numBytesInRemote
	numBytesInRemotePerSecond
	numBuffersInLocal
	numBuffersInLocalPerSecond
	numBuffersInRemote
	numBuffersInRemotePerSecond
	numBytesOut
	numBytesOutPerSecond
	numBuffersOut
	numBuffersOutPerSecond
Task/Operator	numRecordsIn
	numRecordsInPerSecond
	numRecordsOut
	numRecordsOutPerSecond
	numLateRecordsDropped
	currentInputWatermark
Operator	currentInput1Watermark
	currentInput2Watermark
	currentOutputWatermark
	numSplitsProcessed

Tablo 3.10 iş, görev veya operatöre ait tüm giriş çıkış metriklerini göstermektedir.

Tez çalışması kapsamında Apache Flink'in bize sağlayacağı tüm metrikler incelendikten sonra bizi en doğru sonuca ulaştıracak metriklerin bulunmasına çalışılmıştır. Bu metrikleri seçerken birbirini etkileyerek aynı oranda azalıp artan metrikler -tamamıyla birbiriyle korele olan-, ölçeklemeyle ilgisi olmayan, entegrasyonu zor olan metrikler tercih dışı bırakılmıştır**.** Bu çalışmada kullanacağımız metrikler Tablo 3.11'de verilmiştir.

Tablo 3.11: Bu çalışmada kullanılacak metrikler.

Alan	**İç takı**	**Metrik**
Job-/TaskManager	Status.JVM.CPU	Load
Job-/TaskManager	Status.JVM.Memory	Heap.Used
Task/Operator	-	numRecordsInPerSecond
Operator	-	currentProcessingTime
Operator	-	currentOutputWatermark

Tablo 3.11'deki metrikler şunları ifade etmektedir:

- **İşlemci Yükü:** Java Sana Makinesi'nin en son elde edilen işlemci yükünü gösterir.
- **Hafıza Kullanımı:** Bayt cinsinde, mevcut durumda kullanılan *heap* hafıza kullanımı.
- **Saniyede Gelen Kayıt Sayısı:** Saniyelik birim zamanda, seçilen operatör veya göreve ulaşan kayıt sayısı.
- **Mevcut İşleme Zamanı:** Gelen verinin olay zamanından bağımsız işlenme zamanı.
- **Mevcut Filigran Çıktısı Zamanı:** Filigranlar gelen verinin olay zamanında ilerlemesini işaretler. Bu metrik bu işaretleri sağlar.

Bu akan veriyi işleyen dağıtık sistemlerde gecikme olması kaçınılmaz bir durumdur. Sistemin ölçeklenmesi gereken durum hesaplanırken gecikmelerinde göz önüne mutlaka alınması gerekir. Oluşturduğumuz sistem üzerinde gecikmenin sistem üzerindeki etkisi sistemin veriyi işleme süresi ve sistemin bu veri için filigran oluşturması arasındaki zaman farkı olarak ele alınabilir. Bu nedenle sistem metrikleri üzerinde gecikme Mevcut İşleme Zamanı ile Mevcut Filigran Çıktısı Zamanı'nın çıkarılması şeklinde ele alınmıştır.

Kullanacağımız metriklerin birimleri Flink'in ölçeklenmesini tetikleyen veriyi üreteceğinden dolayı büyük önem taşımaktadır. Metriklerin birimleri Tablo 3.12'de tanımlanmıştır.

Tablo 3.12: Çalışmada kullanacağımız metriklerin birimleri.

Metrik	**Birim**
İşlemci Yükü	Oran
Hafıza Kullanımı	Bayt
Kayıt Sayısı	Ölçü (Integer)
Gecikme	Saniye

Metrikleri barındıran veri seti içerisinden örnek veriler Tablo 3.13 gibidir.

Tablo 3.13: Çalışmada kullanılacak veri setinden bir örnek.

Metrik	**Örnek**
İşlemci Yükü	[0.53779959, 0.54670971, 0.57250159, 0.53621445, 0.49351587, 0.49081711]
Hafıza Kullanımı	[2982.4293386, 2962.74074083, 3066.90614069, 3040.31287205, 2970.71550109]
Kayıt Sayısı	[312468, 451728, 114109, 369504, 86553, 162808, 209666, 251185, 382577, 270345]
Gecikme	[8, 20, 26, 11, 18, 26, 6, 7, 1, 18, 29, 8, 30, 20, 14]

Geliştirdiğimiz sistemin ölçeklenmesine karar vermek için çeşitli matematiksel ve istatiksel hesaplamalar yapılmıştır. Bu hesaplamalarda kullanılan sistem değişkenleri ve hesaplamalar Tablo 3.14'de açıklanmıştır.

Tablo 3.14: Çalışmada kullanılan değişken ve sabitler.

Sembol	**İsim**	**Açıklama**	**Formül**
c	İşlemci yükü	Java Sana Makinesi'nin en son elde edilen işlemci yükü.	-
C	İşlemci yükü ortalama değeri	-	$\frac{1}{n}\left(\sum_{i=1}^{n} c_i\right)$
H	Hafıza kullanım oranı	Hafıza kullanımın toplam hafızaya oranıdır.	-

M	Hafıza kullanım oranının ortalama değeri	-	$\frac{1}{n}\left(\sum_{i=1}^{n} H_i\right)$
x	Saniyede gelen kayıt sayısı	Saniyelik birim zamanda, seçilen operatör veya göreve ulaşan kayıt sayısı.	-
X	Saniyede gelen kayıt sayısının ortalama değeri	-	$\frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)$
Y	Saniyede gelen kayıt sayısının standart sapma değeri	-	$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$
R	Saniyede gelen kayıt sayısı eşik oranı	-	$\frac{Y}{X}$
Z	Mevcut işleme zamanı	-	-
T	Mevcut filigran çıktı zamanı	-	-
τ	Gecikme	-	$Z - T$
A	Gecikme süresinin ortalama değeri	-	$\frac{1}{n}\left(\sum_{i=1}^{n} \tau_i\right)$
Q	Gecikme süresi eşik sabiti	Dışarıdan parametre olarak verilir, sabit bir değerdir.	-
B	Gecikme süresi eşik oranı	Eğer A değeri Q değerinden büyükse 0.5, küçükse 0 kabul edilir.	-
J	Lambda değerleri	C, M, R, B değerlerini içeren bir dizidir.	-

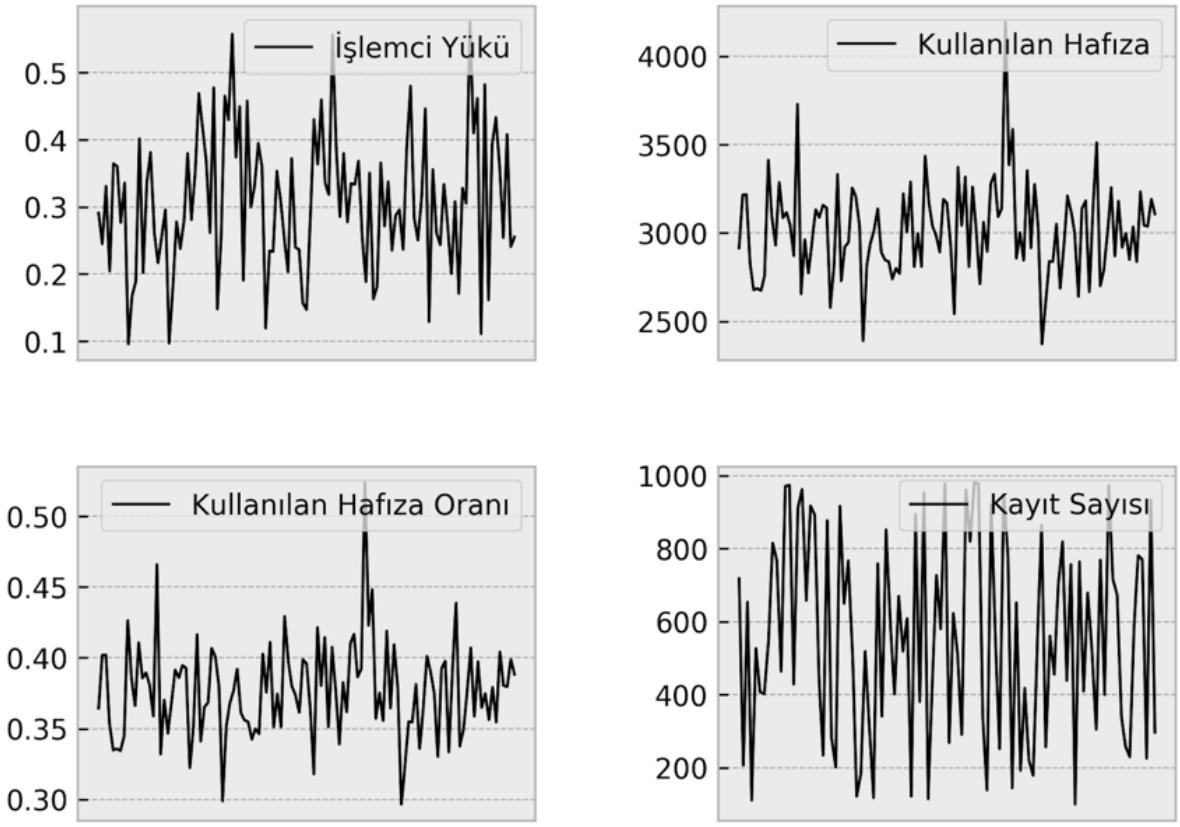
W	Lambda eşik sabiti	Dışarıdan parametre olarak verilir, sistemin ölçeklenmeye geçeceği limiti gösterir. Sabit bir değerdir.	-
λ	Lambda	Lambda değerlerinin ortalama değeridir.	$\frac{1}{n}\left(\sum_{i=1}^{n} J_i\right)$

Tüm bu hesaplamaların sonucunda λ değeri W sabitinden büyükse ölçekleme işlemi tetiklenecektir.

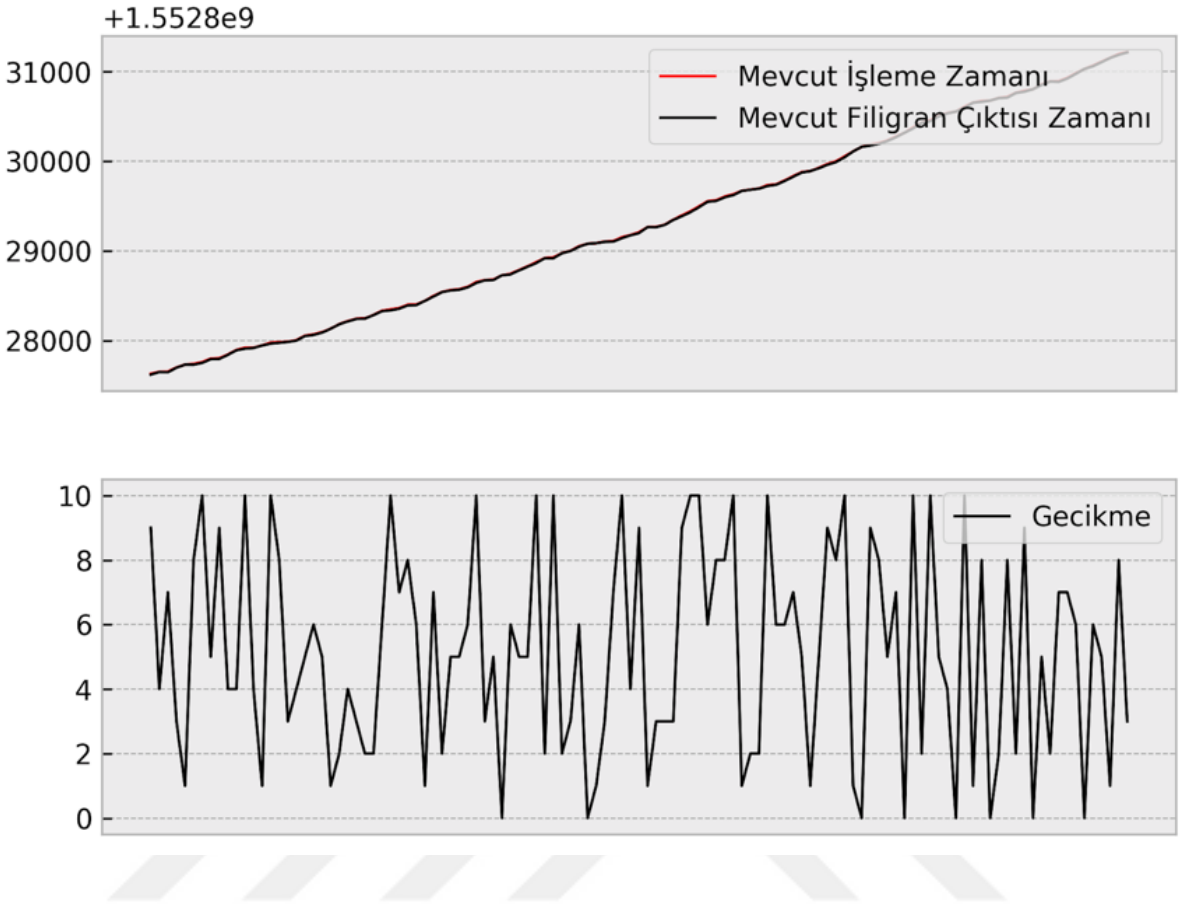
4. BULGULAR

Yapılan deneyde Q sabiti 30, W sabiti 0.6 olarak alınmıştır. Sabit değerler keyfi alınmıştır, Q sabiti gecikmelere karşı sistemin duyarlılığını saniye cinsinden, W sabiti sistemin tüm eşik dengesinin ne kadar olacağına göre belirlenebilir. W sabiti, sıfır ile bir arasında bir değer kabul eder. Örnek veri seti için işlemci yükü, hafıza kullanımı, gelen kayıt sayısı, mevcut işleme zamanı, mevcut çıktı filigranı ve gecikme değerleri çizelgelerdeki gibi olmaktadır. Örnek veri setleri olasılık dağılımları kullanılarak sentetik bir yük üretilerek oluşturulmuştur.

Şekil 4.1'de verinin geliş sırasıyla birinci pencerede gelen işlemci yükü, kullanılan hafıza, kullanılan hafızanın toplam hafızaya oranı ve saniyede gelen kayıt sayısı metrikleri görülmektedir. Şekil 4.2'de sistemin gelen veriyi işleme zamanı ile filigran oluşturduğu zamanı göstermektedir. Daha önce de bahsedildiği gibi ikisi arasındaki fark bize gecikmeyi vermektedir ve bu veri de yine Şekilde 4.2'de gösterilmektedir. Şekil 4.1 ve Şekilde 4.2 çizelgelerini oluşturan ve sisteme giriş olarak verilen girdilerin örnek bir kesiti ise Tablo 4.1'de verilmiştir. Çizelgelerde ve tablolarda bulunan veriler giriş verilerini temsil etmektedir. Tablolarda verinin kesiti gösterilmektedir.



Şekil 4.1: Birinci hesaplama iterasyonu işlemci yükü, kullanılan hafıza, kullanılan hafıza oranı ve kayıt sayısı değerleri.

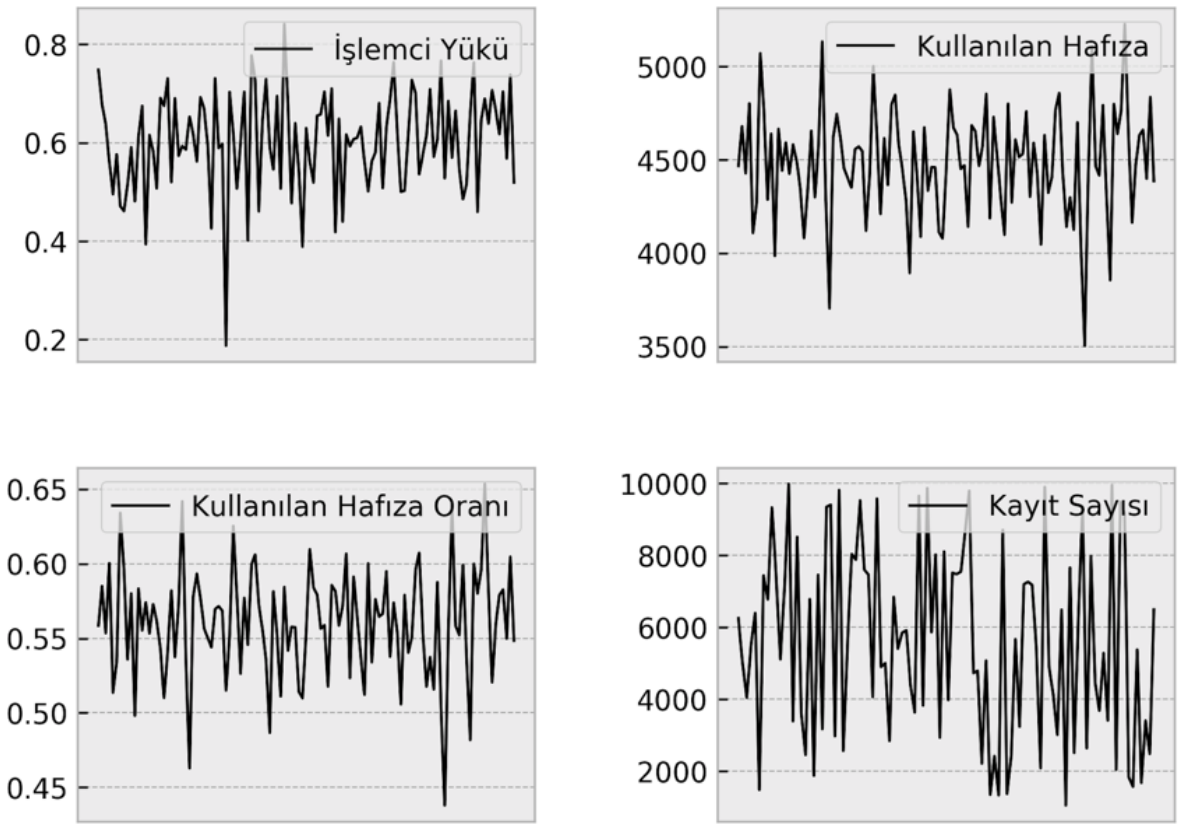


Şekil 4.2: Birinci hesaplama iterasyonu mevcut işleme zamanı, mevcut filigran çıktısı zamanı ve gecikme değerleri.

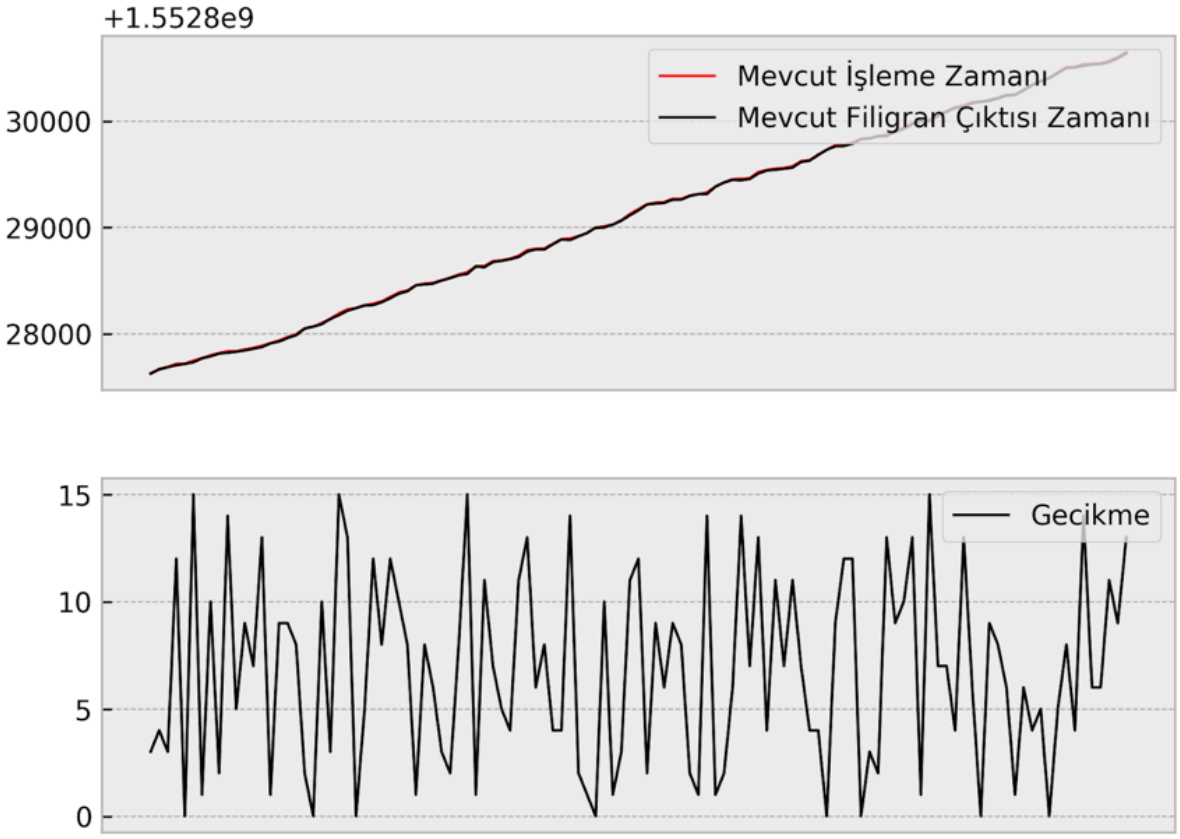
Tablo 4.1: Birinci hesaplama iterasyonu için örnek değerler.

Veri Tipi	**Değer**
İşlemci yükü	[0.29081359, 0.24484373, 0.33099842 0.20422209, 0.36440966, 0.36044093]
Kullanılan hafıza	[2915.05236578, 3215.18824778, 3217.2999415, 2826.96742778]
Kalan hafıza oranı	[0.36438155, 0.40189853, 0.40216249, 0.35337093, 0.33482182, 0.33580286]
Saniyede kalan gelen kayıt sayısı	[718, 207, 654, 111, 527, 408, 402, 548, 815, 767, 464, 971, 974, 429, 912, 962, 658, 917]
Gecikme (τ)	[9, 4, 7, 3, 1, 8, 10, 5, 9, 4, 4, 10, 4, 1, 10, 8, 3, 4, 5, 6, 5, 1, 2, 4, 3, 2, 2, 6, 10]
Lambda değerleri (J)	[0.3075763493045144, 0.37675745680472644, 0.4863931441774576, 0]
λ	0.2926817375716746

λ değeri W sabitinden küçük olduğu için ölçeklemeye ihtiyaç yoktur.



Şekil 4.3: İkinci hesaplama iterasyonu işlemci yükü, kullanılan hafıza, kullanılan hafıza oranı ve kayıt sayısı değerleri.

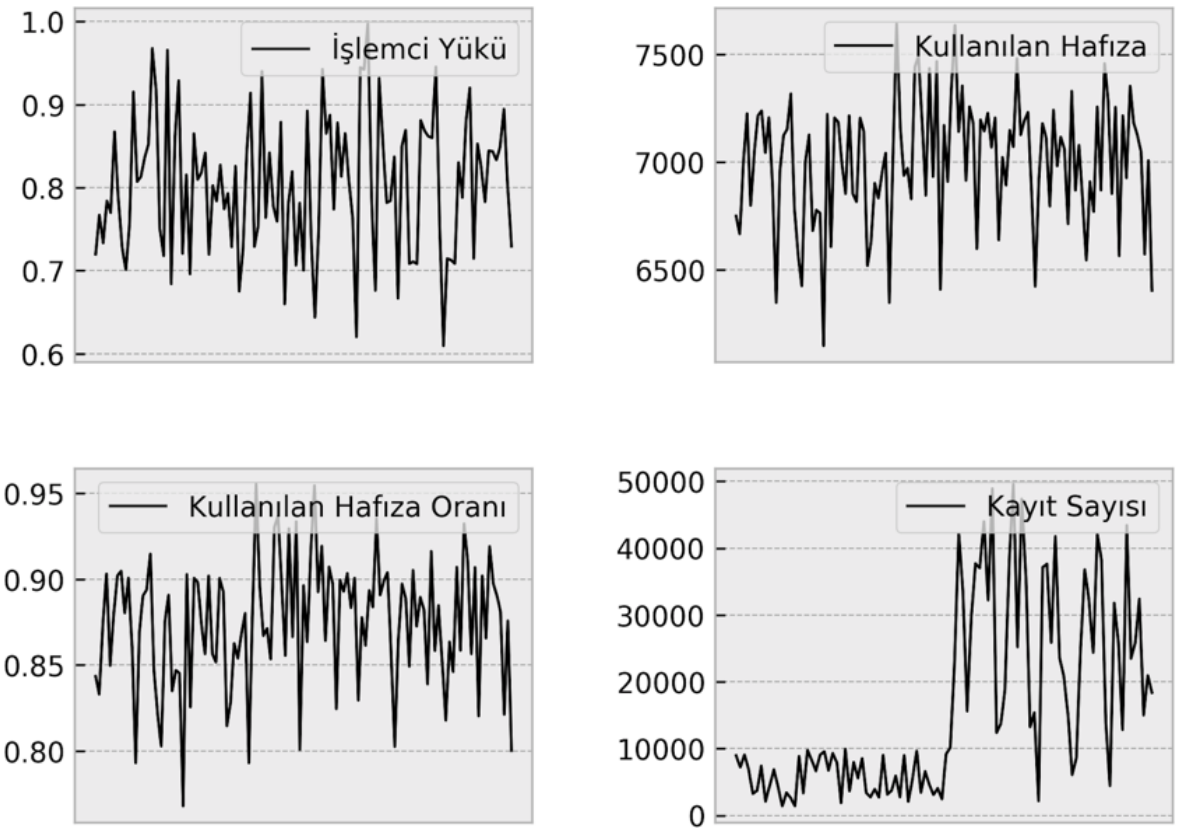


Şekil 4.4: İkinci hesaplama iterasyonu mevcut işleme zamanı, mevcut filigran çıktısı zamanı ve gecikme değerleri.

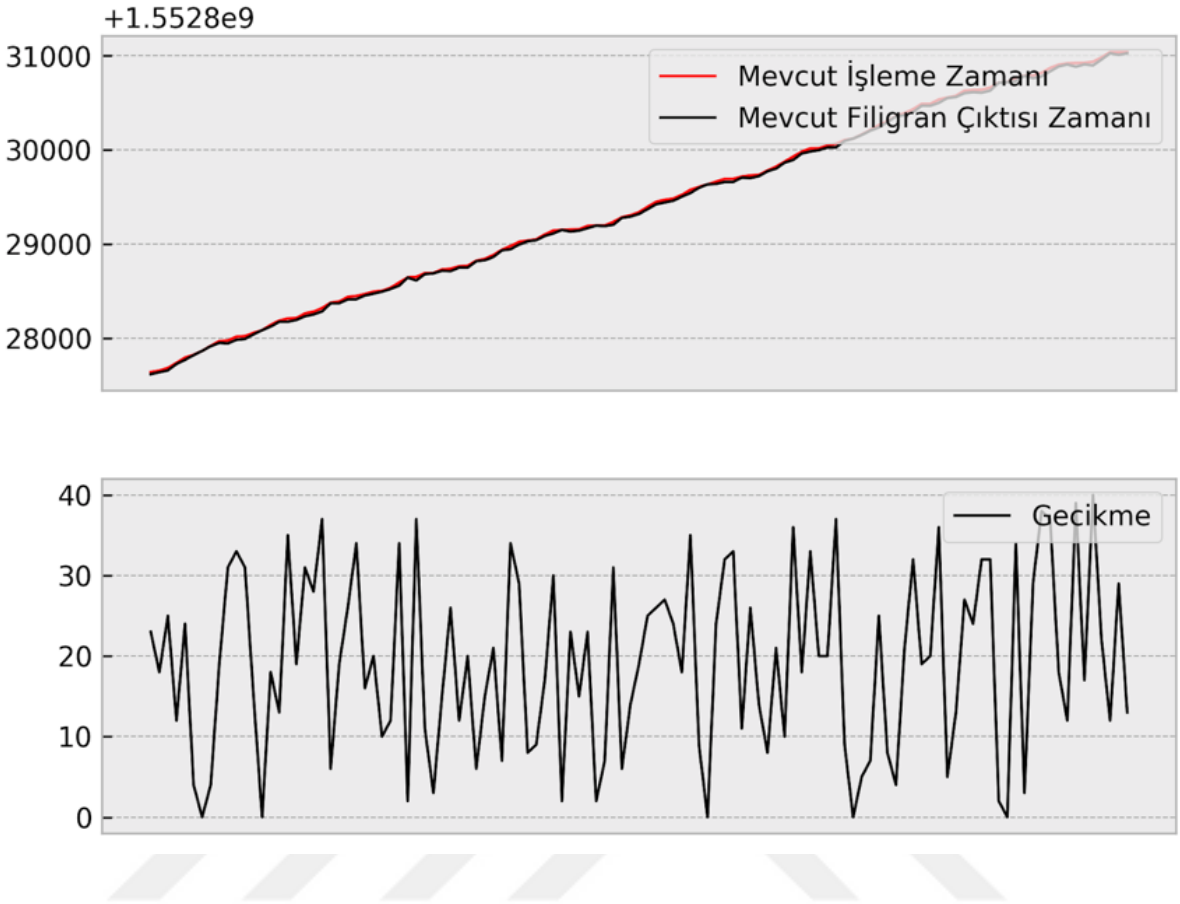
Tablo 4.2: İkinci hesaplama iterasyonu için örnek değerler.

Veri Tipi	**Değer**
İşlemci yükü	[0.74792182, 0.67820917, 0.63737983, 0.56141087, 0.49553547, 0.57619555]
Kullanılan hafıza	[4467.48624105, 4679.77961699, 4427.77417013, 4802.37689798]
Kalan hafıza oranı	[0.55843578, 0.58497245, 0.55347177 0.60029711, 0.51352895, 0.53401568]
Saniyede kalan gelen kayıt sayısı	[6248, 4977, 4042, 5477, 6393, 1481, 7435, 6772, 9323, 7411, 5108, 7106, 9981, 3385]
Gecikme (τ)	[3, 4, 3, 12, 0, 15, 1, 10, 2, 14, 5, 9, 7, 13, 1, 9, 9, 8, 2, 0, 10, 3, 15, 13, 0, 5, 12, 8]
Lambda değerleri (J)	[0.6022923124276031, 0.560108109236605, 0.4695572727535386, 0]
λ	0.4079894236044367

Bir önceki iterasyonda olduğu gibi Şekil 4.3'de verinin geliş sırasıyla ikinci pencerede gelen işlemci yükü, kullanılan hafıza, kullanılan hafızanın toplam hafızaya oranı ve saniyede gelen kayıt sayısı metrikleri görülmektedir. Şekil 4.4'de sistemin gelen veriyi işleme zamanı ile filigran oluşturduğu zamanı ve gecikmeyi göstermektedir. Şekil 4.3 ve Şekilde 4.4 çizelgelerini oluşturan ve sistem giriş olarak verilen girdilerin örnek bir kesiti ise Tablo 4.2'de verilmiştir. Çizelgelere dikkat edilirse birim zamanda gelen kayıt sayısının artışına bağlı olarak, sistemin işlemci yükü ve hafıza kullanımı gözle görülür bir şekilde artmıştır. Fakat Tablo 4.2'de görüleceği üzere λ değeri halen W sabitinden küçük olduğu için ölçeklemeye ihtiyaç yoktur.



Şekil 4.5: Üçüncü hesaplama iterasyonu işlemci yükü, kullanılan hafıza, kullanılan hafıza oranı ve kayıt sayısı değerleri.

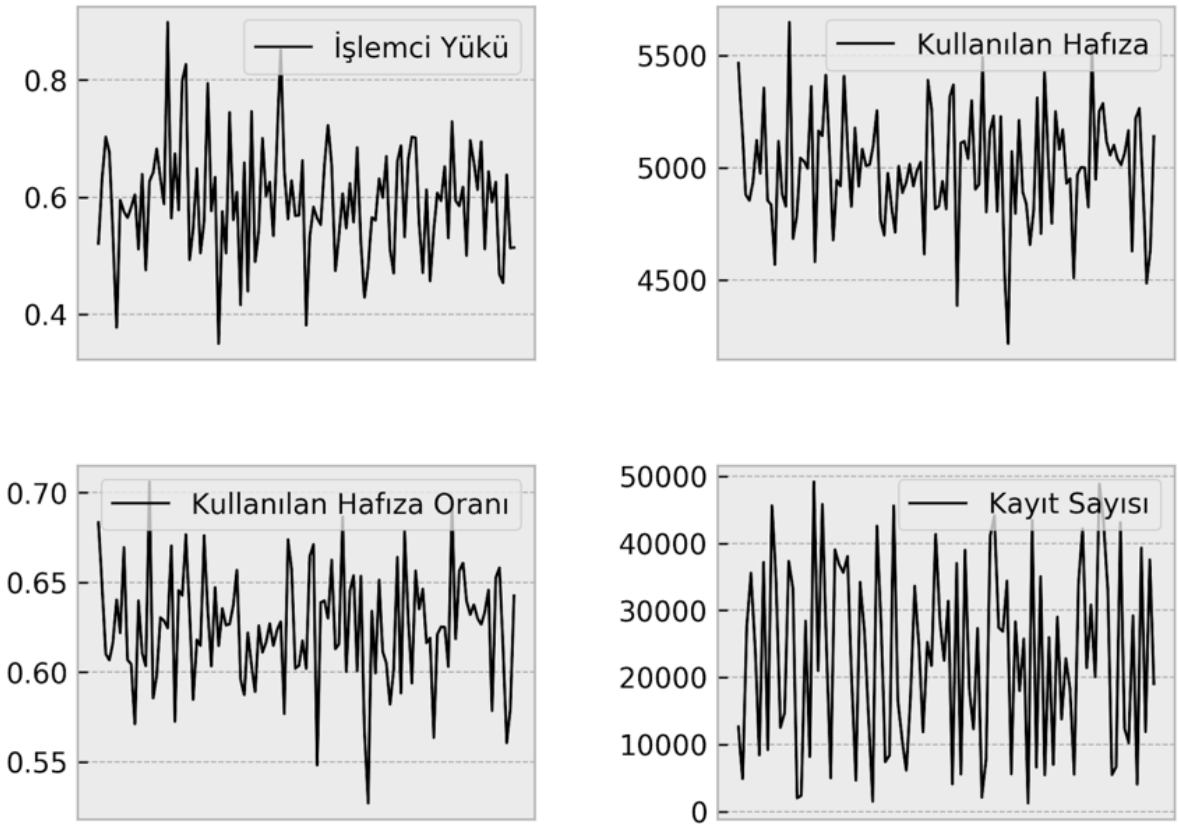


Şekil 4.6: Üçüncü hesaplama iterasyonu mevcut işleme zamanı, mevcut filigran çıktısı zamanı ve gecikme değerleri.

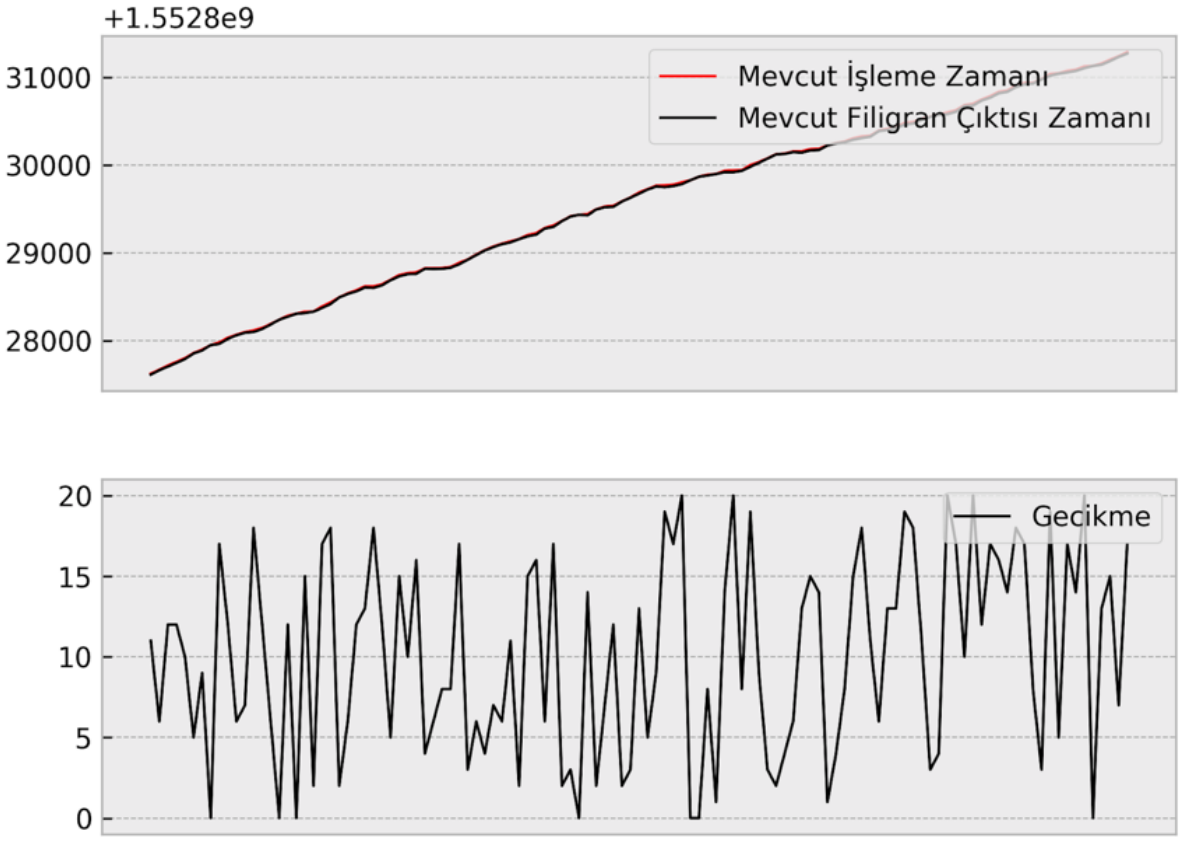
Tablo 4.3: Üçüncü hesaplama iterasyonu için örnek değerler.

Veri Tipi	**Değer**
İşlemci yükü	[0.71975049, 0.76689485, 0.7333771, 0.78396634, 0.76981594, 0.86728982]
Kullanılan hafıza	[6748.09165228, 6664.69131265, 6994.82196045, 7223.84370897]
Kalan hafıza oranı	[0.84351146, 0.83308641, 0.87435275, 0.90298046, 0.84965956, 0.88083314]
Saniyede kalan gelen kayıt sayısı	[8953, 7201, 9061, 6826, 3234, 3706, 7406, 2080, 4420, 6853, 4224, 1388, 3376, 2597]
Gecikme (τ)	[23, 18, 25, 12, 24, 4, 0, 4, 19, 31, 33, 31, 15, 0, 18, 13, 35, 19, 31, 28, 37, 6, 19, 26]
Lambda değerleri (J)	[0.8027545722795377, 0.8746252806617266, 0.8664009685964649, 0]
λ	0.6359452053844323

Bir önceki iterasyona benzer şekilde Şekil 4.5'de verinin geliş sırasıyla üçüncü pencerede gelen işlemci yükü, kullanılan hafıza, kullanılan hafızanın toplam hafızaya oranı ve saniyede gelen kayıt sayısı metrikleri görülmektedir. Şekil 4.6'de sistemin gelen veriyi işleme zamanı ile filigran oluşturduğu zamanı ve gecikmeyi göstermektedir. Şekil 4.5 ve Şekilde 4.6 çizelgelerini oluşturan ve sistem giriş olarak verilen girdilerin örnek bir kesiti ise Tablo 4.3'de verilmiştir. Çizelgelere dikkat edilirse birim zamanda gelen kayıt sayısının anormal bir şekilde artmıştır ve tepe yapmıştır. Bu artışa bağlı olarak, sistemin işlemci yükü ve hafıza kullanımı sınırları zorlayacak şekilde artmıştır. Bunun yanında gecikmeler de birim zamanda gelen kayıt sayısını doğru zamanda işlemek için çaba sarf etmektedir fakat yüksek gecikmeler de oluşmaya başlamıştır. Tablo 4.3'de görüleceği üzere λ değeri bu iterasyonda W sabitinden büyük olduğu için sisteme ölçekleme isteği gönderilir.



Şekil 4.7: Ölçeklemeden sonra işlemci yükü, kullanılan hafıza, kullanılan hafıza oranı ve kayıt sayısı değerleri.

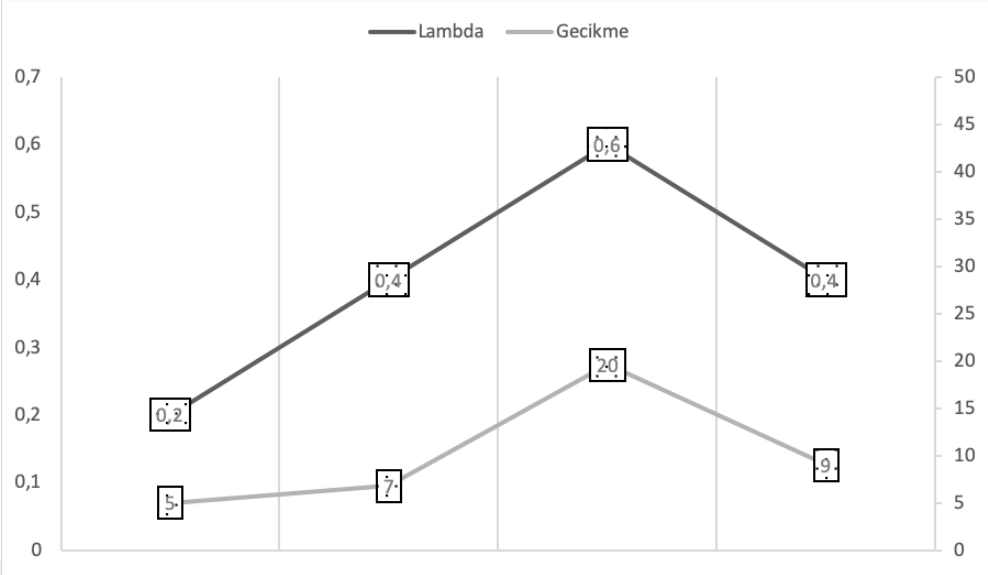


Şekil 4.8: Ölçeklemeden sonra işleme zamanı, mevcut filigran çıktısı zamanı ve gecikme değerleri.

Tablo 4.4: Ölçeklendikten sonra örnek değerler.

Veri Tipi	**Değer**
İşlemci yükü	[0.52061993, 0.62960358, 0.70270973, 0.67735602, 0.53360672, 0.37741859]
Kullanılan hafıza	[5466.4133859, 5189.29183283, 4881.01291935, 4854.00134743]
Kalan hafıza oranı	[0.68330167, 0.64866148, 0.61012661, 0.60675017, 0.61677137, 0.6403014]
Saniyede kalan gelen kayıt sayısı	[12623, 4870, 27608, 35565, 24276, 8399, 37170, 9201, 45611, 34264, 12502, 14661]
Gecikme (τ)	[11, 6, 12, 12, 10, 5, 9, 0, 17, 12, 6, 7, 18, 12, 6, 0, 12, 0, 15, 2, 17, 18, 2, 6, 12, 13]
Lambda değerleri (J)	[0.5935922476790889, 0.6245944830883645, 0.5973252562509304, 0]
λ	0.45387799675459595

Sistem ölçeklendikten sonra sistemin durumu, işlemci yükü, kullanılan hafıza, kullanılan hafıza oranı ve birim zamanda gelen kayıt sayısı Şekil 4.7'deki gibidir. Şekil 4.7'de görüldüğü gibi birim zamanda gelen kayıt sayısı aynı davranışı göstermektedir. Şekil 4.7'de dikkat edilmesi gereken birim zamanda gelen kayıt sayısına rağmen belirli bir süre sonra işlemci yükü, kullanılan hafıza ve kullanılan hafıza oranı düşmektedir. Bu duruma bağlı olarak da Şekil 4.8'de görüldüğü gibi gecikmeler kabul edilebilir bir noktaya gelmiştir. Şekil 4.7 ve Şekilde 4.8 çizelgelerini oluşturan ve sistemin mevcut durumdaki metriklerinin örnek bir kesiti ise Tablo 4.4'de verilmiştir.



Şekil 4.9: Lambda ve gecikme arasındaki ilişki grafiği.

Şekil 4.9'da λ değeri ile gecikme arasındaki ilişki gösterilmiştir. Şekilde de görüldüğü gibi 0.6 değerine ulaşan λ değeri, ölçekleme işlemi gerçekleştiği için 0.4 değerine düşmüştür. λ 'nın bu değişimine korele bir şekilde gecikmeler artmakta ve azalmaktadır.

Ölçekleme isteği gönderildikten sonra Apache Flink kümesi λ değerinin küçülmesi ile birlikte işlemci kullanımı normal seviyelere gelmiş, hafızayı daha etkin bir şekilde kullanmaya başlamış ve birim zamanda gelen kayıt sayısı artmasına rağmen sistem işleme zamanı, olayın gerçekleşme zamanına yaklaşmıştır. Bunun sonucunda hem kaynak kullanımı optimize edilmiş hem gecikmelerin önüne geçilmiştir.

5. TARTIŞMA VE SONUÇ

Kurumlar her geçen gün daha hızlı bir şekilde veri üretmektedir. Hızla artan veri kaynakları birçok veri akışı oluşturmaktadır. Kurumlar müşterileri, uygulamaları ve ürünleri daha iyi tanımak ve anlamak için bu akan veriyi işleme ve analiz etme ihtiyacı duymaktadır. Yığın halinde verinin analiz edilmesine göre, gerçek zamanlı veriyi işlemek çok daha hızlı karar alınmasını ve günün şartlarına hızlı adapte olabilmeyi sağlar. Fakat gerçek zamanlı akan veriyi işleyen sistemleri inşa etmek ve operasyonunu sağlamak oldukça zor bir iştir. Maliyeti de göz önünde bulundurarak sistemin gerçek zamanlı bir şekilde çalışmaya devam edebilmesi için dağıtık sunucuların yönetilebilmesi ve gerektiğinde kolayca ölçeklenebilmesi gerekmektedir.

Bu tez çalışmasında akan veriyi işleyen dağıtık sistemlerin, değişen iş yüklerine adapte olması adına ölçeklenebilirlik problemlerine odaklanarak, Apache Flink üzerinde bu işi yapabilen bir sistem önerilmektedir. Önerilen sistemde tüketen tarafın kaynak ve çalışan iş özelindeki metrikler kullanılarak sistemin o an ölçeklenmesi gerekip gerekmediğine karar verilmesine olanak sağlanmıştır. Bunun yanında bu sürecin daha soyut bir şekilde yapılması ve farklı akan veriyi işleyen dağıtık sistemlere de kolay entegre olabilmesi amaçlanmıştır.

Sonuç olarak akan veriyi işleyen dağıtık sistemlerde kullanılabilecek bir sistem önerisinde bulunularak, bu sistemin etkin bir şekilde çalıştığı çeşitli simülasyonlarla desteklenerek gösterilmiştir. Literatürde akan veri üzerine çeşitli çalışmalar vardır. Literatürde akan veriyi işleyen dağıtık sistemler üzerine çeşitli çalışmalar vardır. Bu çalışmalarda sistemin değişen iş yüklerine adapte olabilmesi ve ölçeklenebilirlikten çok sistemin olağan şartlarda nasıl çalışacağına odaklanmıştır. Ölçeklenebilirlik üzerine çok kısıtlı sayıda çalışma vardır. Özellikle de Apache Flink üzerine yapılan çalışma sayısı çok daha kısıtlıdır. Önerilen bu sistem literatürde diğer yapılan çalışmalara özellikle de ölçeklenebilirlik açısından alternatif bir yapı olarak kullanılabilir.

Gelecek çalışmalarda yapılabilecek çeşitli planlar mevcuttur. Bunlardan ilki hesap yapılan metrik verilerini özelleştirip türeterek metrik sayısını arttırmaktır. İkinci plan, üreten tarafın da metriklerini hesaba katmaktır. Üreten taraf akan veri işleyen dağıtık sistem veriyi gönderen birimdir. Bu bir Apache Kafka gibi bir mesaj kuyruğu olabilir. Üreten tarafında metrikleri hesaba katılırsa nasıl bir hacimde sisteme veri akacağı öngörülebilir. Üçüncü plan ise metrik verilerinin belirli bir matematiksel ve istatistiksel hesap yerine, makine öğrenmesi algoritmaları

kullanılarak daha etkin bir ölçekleme stratejisi oluşturmaktır. Bundan sonraki aşamada, mevcut çalışmaya planlar dahilinde belirlenen yeni özellikler sırasıyla eklenerek sistemin geliştirilmesi üzerine çalışılacaktır.

KAYNAKLAR

[1]. Kombi R., Lumineau N., Lamarre P., 2017, A preventive auto-parallelization approach for elastic stream processing, IEEE 37th International Conference on Distributed Computing Systems

[2]. HoseinyFarahabady R., Samani H., Wang Y., Zomaya A., Tari Z., 2016, A QoS-Aware Controller for Apache Storm, IEEE 15th International Symposium on Network Computing and Applications

[3]. Renart E., Diaz-Montes J., Parashar M., 2017, Data-driven Stream Processing at the Edge, IEEE 1st International Conference on Fog and Edge Computing (ICFEC)

[4]. Papageorgiou A., Poormohammady E., Cheng B., 2016, Edge-Computing-aware Deployment of Stream Processing Tasks based on Topology-external Information: Model, Algorithms, and a Storm-based Prototype, IEEE International Congress on Big Data

[5]. De Matteis T., Mencagli G., 2017, Elastic Scaling for Distributed Latency-sensitive Data Stream Operators, 25th Euromicro International Conference on Parallel, Distributed and Network-Based Processing

[6]. Basanta-Val. P., Garcia N., Fernandez L., Fiesteus J., 2017, Patterns for Distributed Real-Time Stream Processing, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS

[7]. Xu L., Peng B., Gupta I., 2016, Stela: Enabling Stream Processing Systems to Scale-in and Scale-out On-demand, IEEE International Conference on Cloud Engineering

[8]. Zhang J., Li C., Zhu L., Liu Y., 2016, The Real-time Scheduling Strategy Based on Traffic and Load Balancing in Storm, IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems

[9]. Chakraborty R., Majumdar S., A Priority Based Resource Scheduling Technique for Multitenant Storm Clusters, 2016, International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)

[10]. Wang C., Meng X., Guo Q., Weng Z., Yang C., 2017, Automating Characterization Deployment in Distributed Data Stream Management Systems, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING

[11]. KHOSHKBARFOROUSHHA A., Ranjan R., Gaire R., ABBASNEJAD E., Wang L., Zomaya A., 2016, Distribution Based Workload Modelling of Continuous Queries in Clouds, IEEE Transactions on Emerging Topics in Computing

[12]. Li T., Tang J., Xu J., 2016, Performance Modeling and Predictive Scheduling for Distributed Stream Data Processing, IEEE TRANSACTIONS ON BIG DATA

[13]. Runsewe O., Samaan N., 2017, Cloud Resource Scaling for Big Data Streaming Applications Using A Layered Multi-dimensional Hidden Markov Model, 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

[14]. Buddhika T., Stern R., Lindburg K., Ericson K., Pallickara S., 2017, Online Scheduling and Interference Alleviation for Low-Latency, High-Throughput Processing of Data Streams, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS

[15]. HoseinyFarahabady M., Zomaya A., Tari Z., 2017, QoS- and Contention- Aware Resource Provisioning in a Stream Processing Engine, IEEE International Conference on Cluster Computing

[16]. Qian W., Shen Q., Qin J., Yang D., Yang Y., Wu Z., 2016, S-Storm: A Slot-aware Scheduling Strategy for Even Scheduler in Storm, IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems

[17]. Chen H., Zhang F., Jin H., 2017, Popularity-aware Differentiated Distributed Stream Processing on Skewed Streams, IEEE 25th International Conference on Network Protocols

[18]. Liu X., Buyya R., 2019, Performance-Oriented Deployment of Streaming Applications on Cloud, IEEE Transactions on Big Data

[19]. Wang Y., Tari Z., HoseinyFarahabady M., Zomaya A., 2017, QoS-aware resource allocation for stream processing engines using priority channels, IEEE 16th International Symposium on Network Computing and Applications

[20]. Apache Hadoop YARN, https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html [Ziyaret Tarihi: 21 Nisan 2019]

[21]. What is Kubernetes, https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ [Ziyaret Tarihi: 21 Nisan 2019]

[22]. Mesos Architecture, http://mesos.apache.org/documentation/latest/architecture/ [Ziyaret Tarihi: 21 Nisan 2019]

ÖZGEÇMİŞ

Kişisel Bilgiler	
Adı Soyadı	Mert KAVİ
Doğum Yeri	Antakya
Doğum Tarihi	02.11.1991
Uyruğu	☑ T.C. ❒ Diğer:
Telefon	5396299209
E-Posta Adresi	mertkavi@gmail.com
Web Adresi	www.mertkavi.com

Eğitim Bilgileri	
Lisans	
Üniversite	Erciyes Üniversitesi
Fakülte	Mühendislik Fakültesi
Bölümü	Bilgisayar Mühendisliği
Mezuniyet Yılı	23.06.2015

Yüksek Lisans	
Üniversite	İstanbul Üniversitesi-Cerrahpaşa
Enstitü Adı	Lisansüstü Eğitim Enstitüsü
Anabilim Dalı	Bilgisayar Mühendisliği Anabilim Dalı
Programı	Program Adı

Makale ve Bildiriler
Kavi M., Orman Z., "Dynamic Scaling At Distributed Data Stream Processing Systems", 1st International Symposium on Graduate Research in Science: Entrepreneurship and Innovation (ISGRS2018), İSTANBUL, TÜRKIYE, 4-6 Ekim 2018.