İLERİ MÜHENDİSLİK ÇALIŞMALARI VE TEKNOLOJİLERİ DERGİSİ

Akan Veri İşleyen Dağıtık Sistemlerde Gecikme Duyarlı Dinamik Ölçekleme İçin Bir Sistem Tasarımı

Zeynep ORMAN*[1], Mert KAVİ*[2]

İstanbul Üniversitesi-Cerrahpaşa, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, 34320, İstanbul

Araştırma Makalesi, Geliş Tarihi: 13.07.2020, Kabul Tarihi: 11.08.2020

Özet

Büyük ölçekli akan veri işleyen dağıtık sistemleri inşa etmek ve operasyonunu sağlamak oldukça karmaşık ve maliyetli bir süreçtir. Sistemlerin veri akışının değişen hızlarına adapte olabilmesi ve gerektiğinde ölçeklenebilmesi gerekmektedir. Bu nedenle, akan veriyi işleyen dağıtık sistemlere entegre edilecek etkin bir otomatik ölçekleme sistemi kullanılması çoğu zaman kaçınılmazdır. Son yıllarda, hızla artan akan veri kaynaklarını işleyebilen sistemlere olan ilgi oldukça artmıştır ve literatürde bu alanda yapılan çok sayıda çalışma bulunmaktadır. Ancak bu çalışmaların çoğu sistemin değişen iş yüklerine adapte olabilmesi ve ölçeklenebilirlik konusu yerine sistemin olağan şartlarda nasıl çalışacağı üzerine yoğunlaşmıştır. Literatürde az sayıda olan ölçeklenebilirlik ile ilgili çalışmalarda ise genellikle ölçeklenebilirlik bir kaynak kümesi ile gerçeklenmektedir. Bu çalışmada, literatürdeki bu eksikliklerden yola çıkılarak, Apache Flink üzerinde çalışan, değişen çalışma yüklerine adapte olabilen bir sistem tasarımı önerilmiştir. Apache Flink, hem sistem geliştirme hem de ölçekleme metriklerini hesaplama amacıyla kullanılmıştır. Ölçekleme, Kuyruk Teorisi kullanılarak hesaplanan, sistemde meydana gelmesi beklenen gecikme ve kritik sistem metrikleri değerlendirilerek gerçekleştirilmiştir. Büyük veri işleyen sistemlere entegre çalışabilecek bu model ile sistem performanslarının geliştirilmesi ve kalite kayıplarının azaltılması hedeflenmiştir. Son olarak, sistemin hangi durumlarda ölçeklendiği ve ölçeklemeden sonraki durumu benzetim çalışmaları ile gerçeklenerek önerilen sistemin etkinliği gösterilmiştir.

Anahtar Kelimeler: Dağıtık sistemler, Büyük veri, Akan veri işleme, Ölçeklenebilirlik, Kuyruk teorisi.

A System Design for Latency Aware Dynamic Scaling at Distributed Data Stream Processing System

Abstract

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 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 with Queuing Theory 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.

Keywords: Distributed systems, Big data, Stream processing, Scalability, Queuing theory.

[1]Sorumlu yazar ormanz@istanbul.edu.tr, [2]mertkavi@gmail.com

1. GİRİŞ

Gelişen teknoloji ile birlikte kuruluşlar büyük hacimde ve hızda veri üretmektedir. Sayısı sürekli artan veri kaynakları birçok veri akışı oluşturmaktadır. Uygulama sunucularından gelen log verileri, web sitelerinden, mobil uygulamalardan gelen klik akışları, Nesnelerin İnterneti'ni oluşturan aygıtlardan gelen ölçüm verilerinin tümü, müşterileri, uygulamaları ve ürünleri daha iyi anlamak, tanımak ve analiz etmek için yardımcı olmaktadır. Oluşan bu veri akışlarını gerçek zamanlı olarak işleyebilmek ve analiz edebilmek, uygulamaları devamlı olarak gözlemleyebilmek ve bu sayede müşterilerine kişiselleştirilmiş teklifler vererek ürün önerileri yapabilmek kurumlara rekabet ortamında büyük avantajlar sağlamaktadır. Web site analitiği ve makine öğrenmesi 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 yığın analitikten akan analitiğe geçiş ve gerçek zamanlı uygulamalar inşa edilmesi şeklinde iki tür kullanım senaryosu vardır. Yığın veriyi işleme ve akan veriyi işleme arasında bazı temel farklılıklar vardır. Akan veriyi toplamak, hazırlamak ve işlemek için, yığın veriyi işlemeye göre farklı araçlar kullanılır. Yığın veri işleyen geleneksel sistemlerde 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. 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 gerektiğinde ölçeklenmesi kaçınılmazdır. Tüm bu işlemlerin verimsiz 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.

Dağıtık akan veri işleme sistemleri son yıllarda oldukça popülarite kazanmış ve literatürde üzerine çok sayıda çalışma yapılmış bir konudur. Bu çalışmaların bir kısmında dağıtık sistemleri kullanarak gerçek zamanlı veriyi analiz edebilmek ve sistem ölçekleyebilmek gibi problemler de ele alınmıştır. (Kombi, Lumineau ve Lamarre, 2017; Farahabady, Samani, Wang, Zomaya ve Tari, 2016; Renart, Diaz-Montes ve Parashar, 2017; Papageorgiou, Poormohammady ve Cheng, 2016; De Matteis ve Mencagli, 2017; Basanta-Val, Garcia, Fernandez ve Fiesteus, 2017; Xu, Peng ve Gupta, 2016; Zhang, Li, Zhu ve Liu, 2016) çalışmalarında Apache Storm üzerinde operatör tıkanıklıkları ve gecikmeler göz önüne alınarak ele alınan sistemlerin ölçeklenmesi amaçlanmıştır. Bu çalışmalarda geçekleştirilen işlemlerin ayrıntısı ise şu şekildedir: (Kombi, Lumineau ve Lamarre, 2017) 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 verimlilik analizi gerçekleştirilmiştir. (Farahabady, Samani, Wang, Zomaya ve Tari, 2016) çalışmasında "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 kalitesini düşürmeden, kaynak kullanımı oranlarına bakılarak gösterilmiştir. (Renart, Diaz-Montes ve Parashar, 2017)'de kullanıcılara programatik olarak iş akışı tasarlanmasına olanak sağlanmıştır ve sistemin akan verinin içeriğine göre reaktif bir davranış sergilemesi amaçlanmıştır. Çalışmada, belirlenen konumlardaki kaynakların arttırılarak bir kaldıraç etkisi yaratılması yaklaşımı ele alınmıştır. Bu şekilde gecikmelerin azaltılması sağlanmıştır. Bu yaklaşımın, tek bir konum ile birden fazla konumun karşılaştırılması yapılarak verimlilik sağladığı gösterilmiştir. (Papageorgiou, Poormohammady ve Cheng, 2016) çalışmasında etkileşim, veritabanı, en son gerçekleştirilen görevleri toplayan bir modül, karar verici algoritmayı sağlayan bir modül ve topolojiyi güncelleyen bir modülden oluşan entegre bir çatı tasarlanmıştır. Tasarlanan algoritma kural tabanlı bir yapıda çalışmaktadır. Görev örneklemleri sayıları karşılaştırılarak verimlilik

gösterilmiştir. (De Matteis ve Mencagli, 2017) çalışmasında servis kalitesini bozmadan, daha düşük maliyetli çözümler bulunması amaçlanmıştır. Tahmini model temelli bir yaklaşım kullanılarak, sentetik ve gerçek veri kümeleri ile gecikmeler gösterilmiştir. (Basanta-Val, Garcia, Fernandez ve Fiesteus, 2017) çalışmasında, 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. (Xu, Peng ve Gupta, 2016) çalışmasında Storm topolojisindeki operatörlerin giriş oranlarına bakılarak bir tıkanıklık metriği hesaplanmıştır. Tıkanıklığın belli bir eşik değerinin üstüne çıkması halinde ölçekleme gerçekleştirilmiştir. Önerilen sistem, farklı topoloji tipleri üzerinde denenerek, birim zamanda ortalama çıktı sayısına göre verimlilik gösterilmiştir. (Zhang, Li, Zhu ve Liu, 2016) çalışmasında, düğümler arasındaki iç trafik ve işçi düğümler arasındaki etkili olmayan yük dengeleri gözönüne alınarak 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 durumları gösterilmiştir.

Başka bir çalışmada, Apache Storm üzerinde statik bir kaynak konfigürasyonu ile topolojilerin önceliklendirilmesi ile sistem çıktısının nasıl arttırılabileceği üzerine çalışılmıştır (Chakraborty ve Majumdar, 2016). (Wang, Meng, Guo, Weng ve Yang, 2017; Khoshkbarforoushha, Ranjan, Gaire, Abbasnejad ve Wang, 2016; Li, Tang ve Xu, 2016) çalışmalarında Apache Storm üzerinde makine öğrenmesi yaklaşımları kullanılarak kaynakların verimliliğinin arttırılması hedeflenmiştir. Bu çalışmalarda geçekleştirilen işlemlerin ayrıntısı ise şu şekildedir: (Wang, Meng, Guo, Weng ve Yang, 2017) 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 eğitilen bir model kullanılarak yeni bir iş yükü geldiğinde ne kadar kaynak tüketileceği makine öğrenmesi yöntemleri ile tahmin edilmiştir. Bu yaklaşım SPS-Storm ile test edilmiştir. (Khoshkbarforoushha, Ranjan, Gaire, Abbasnejad ve Wang, 2016)'da olasılık yoğunluğu fonksiyonları kullanılarak, kaynak kullanımı için bir gösterge oluşturması amaçlanmıştır. (Wang, Meng, Guo, Weng ve Yang, 2017) çalışmasında olduğu gibi burada da geçmişe dayalı sistem kayıtları göz önüne alınarak makine öğrenmesi teknikleri ile model eğitilmiştir. Bu amaçla Yapay Sinir Ağları'nın özel bir türü olan Karışık Yoğunluk Ağları kullanılmıştır. Tahmin edilen ve gerçekte olan işlemci yükü ve hafıza kullanımı oranlarına bakılarak verimlilik gösterilmiştir. (Li, Tang ve Xu, 2016) çalışmasında ortalama çokuzlu işleme zamanı tahmin edilmek istenmiştir. İş parçacıkları arasındaki gecikme temel ağırlıklardan biri olarak alınmıştır. Eğitimli öğrenme algoritması kullanılarak, işlemci yükü, hafıza, iş parçacığı yükü gibi özellikler ile sistemin bir öğrenme sürecinden geçmesi sağlanmıştır. Kelime sayma iş yükü kullanılarak topoloji oluşturulmuştur ve ortalama çokuzlu işleme zamanı ile sistemin verimliliği gösterilmiştir.

(Runsewe ve Samaan, 2017) çalışmasında Hidden-Markov modeli kullanılarak değişen yüklere adapte olabilen bir sistem tasarımı önerilmiştir. (Buddhika, Stern, Lindburg, Ericson ve Pallickara, 2017; HoseinyFarahabady, Zomaya ve Tari, 2017; Qian, Shen, Qin, Yang D., Yang Y. ve Wu, 2016) çalışmalarında ise kaynakların kullanımına göre işlerin önceliklendirilmesine dayalı bir sistem tasarımı amaçlanmıştır. Bu çalışmalarda geçekleştirilen işlemlerin ayrıntısı ise şu şekildedir: (Buddhika, Stern, Lindburg, Ericson ve Pallickara, 2017) çalışmasında geliştirilen algoritma gözlemleme, analiz, planlama ve yürütme olarak dört fazdan oluşmaktadır. Duruma müdahale etmek için çeşitli hesaplamalar yapılmıştır ve bu hesaplamalara "tahmin yüzükleri" adı verilmiştir. Bu yüzükler, analiz ve planlama aşamalarında kullanılmaktadır. Çalışmada, mesaj başına işleme zamanı, mesaj boyutu, bant genişliği kullanımı gibi veriler kullanılmış ve işlemci ve hafıza kullanımları karşılaştırılarak diğer sistemlere göre önerilen sistemin avantajları gösterilmiştir. (HoseinyFarahabady, Zomaya ve Tari, 2017) çalışması konuya bir optimizasyon problemi olarak ele almıştır ve 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 gibi hedefleri mevcuttur. Çalışma sonucunda önerilen sistem ile birlikte kümenin kaynak kullanımı artmış, ortalama çokuzlu gecikmeler ve servisin kalitesini bozacak olaylar azalmıştır. (Chen, Zhang ve Jin, 2017) çalışmasında Apache Storm kaynaklarıyla daha verimli çalışmak için Apache Zookeeper metrikleri göz önüne alınarak alternatif bir önceliklendirme sistem tasarımı geliştirilmiştir. (Liu ve Buyya, 2017; Wang,

Tari, HoseinyFarahabady ve Zomaya, 2017) çalışmalarında işlenen veri seti üzerinde, verinin anahtarlandığı durumlarda, anahtarlamayı göz önüne alarak popüler anahtarları önceliklendirerek sistemin değişken durumlara adapte olabilmesi amaçlanmıştır. Bu çalışmalarda geçekleştirilen işlemlerin ayrıntısı ise şu şekildedir: (Liu ve Buyya, 2017) 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. Sonuç olarak, literatürdeki bu çalışmalar ve benzer çalışmaların birçoğunun akan veri sistemlerini Apache Storm platformu üzerinde gerçekleştirildiği ve ölçeklendirme probleminin farklı yöntemler ve metrikler kullanılarak ele alındığı gözlemlenmiştir.

Özetle, her geçen gün hızla artan akan veri kaynaklarını işleyebilmek büyük önem kazanmıştır. Akan veriyi işleyen sistemler veriyi gerçek zamanlı olarak işlediği için sistem çok çeşitli senaryolara adapte olabilmelidir. Kaynakların optimize edilip, çıktının arttırılması temel amaçtır ve akan veriyi işleyen dağıtık sistemlerde dinamik bir ölçeklemeye gereksinim vardır. Bu çalışmada bu gereksinimler göz önüne alınarak Apache Flink üzerinde dinamik ölçeklemeye olanak sağlayan, entegre olabilir bir harici sistem oluşturulması hedeflenmiştir. Bu hedef doğrultusunda akan veri işleyen dağıtık bir sistemin metrikleri Java Management Extensions aracılığıyla, Akka kullanılarak Kafka üzerinde toplanmış ve Apache Flink kullanılarak, çeşitli hesaplamalarla sistemde meydana gelmesi beklenen gecikmede göz önüne alınarak sistemin ölçeklenmesi gerektiği durumlar belirlenmiştir. Aynı zamanda, Kuyruk Teorisi kullanılarak belirlenen veri trafik yoğunluğuna bağlı olarak veri hacminin arttırılıp azaltılabildiği gecikmeye duyarlı bir model önerilmiştir. Literatürde yapılan daha önceki çalışmalardan farklı olarak önerilen sistem, akan veri işleyen dağıtık sistemlerden bağımsız, soyut bir şekilde birçok sisteme entegre olabilecek ve ölçekleme işlemini Apache Flink üzerinde gerçekleştirebilecek şekilde tasarlanmıştır. Aynı zamanda bağımsız metrik toplama modülüne de sahiptir. Bu sistemin en önemli özelliklerinden bir tanesi de 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 model hem performans hem de maliyet açısından dağıtık akan veri işleyen bir sistemi olduğundan daha verimli çalışan bir sistem haline getirmektedir.

Çalışmanın ilerleyen bölümleri şu şekilde organize edilmiştir: İkinci bölümde, dağıtık akan veriyi işlemek için kullanılan platformlar ve önerilen sistem mimarisi, bu mimariyi oluşturan bileşenler ve nasıl bir yöntemle ölçeklemenin gerçekleştirileceği ele alınmıştır. Üçüncü bölümde, akan veriyi işleyen dağıtık bir sistemin benzetim çalışması yapılarak sistemin hangi durumlarda ölçeklendiği ve ölçeklemeden sonraki durumları analiz edilmiştir. Dördüncü bölümde ise sonuç ve gelecekte yapılması planlanan çalışmalardan bahsedilmiştir.

2. ANALİZ METODU

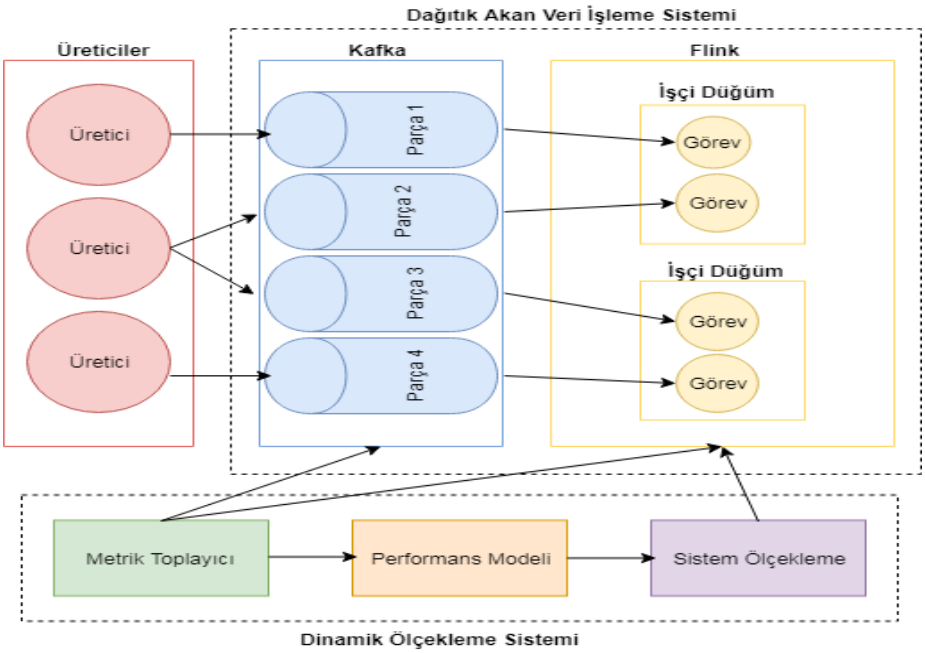
Bu çalışmada, Apache Flink hem geliştirme yapılacak sistem hem de ölçekleme metriklerini alıp hesaplama yapan bir bileşen olarak kullanılmıştır. Akka, Apache Flink metriklerini Apache Kafka'ya gönderen olarak konumlanmıştır. Apache Kafka ise sistem metriklerini toplayan ve dağıtan bir mesaj kuyruğu olarak kullanılmıştır.

Apache Flink açık kaynaklı bir dağıtık akan veri işleme çatısıdır. Akan veriyi işleyen çekirdek dağıtık sistem Java ve Scala ile geliştirilmiştir. Apache Flink düşük gecikmelerle yüksek verimlilik sağlar ve hataları tolere edebilir. Apache Kafka ise dağıtık bir akış platformudur. Akışların kayıtlarını yayınlayabilir ve bu özelliği ile bir mesaj kuyruğuna veya kurumsal mesajlaşma sistemine benzemektedir. Hataları tolere edebilir ve kalıcı bir şekilde akışların kayıtlarını saklayabilir.

2.1. Önerilen Sistem Modeli

Bu bölümde, çalışma kapsamında önerilen ve Şekil 1'de gösterilen sistem mimarisi ve ölçekleme modeli anlatılmaktadır. Oluşturulan bu yapı, literatürdeki mevcut modellere dayanmasının yanı sıra kullanılan yeni formülasyonlar ve yöntemlerle bu modelleri geliştirmektedir.

Şekil 1'de dağıtık Akan Veri İşleme Sistemi ile temsil edilen kısım ölçeklenecek sistemi göstermektedir. Burada işlenecek mesajların bulunduğu bir Kafka kümesi ve bu mesajları işleyen bir Apache Flink kümesi bulunmaktadır.



Şekil 1. Çalışma kapsamında önerilen sistem modeli

Tablo 1. Çalışmada kullanılan sistem metrikleri ve sabitler

Sembol	İsim	Formül	Açıklama
C	İşlemci yükü ortalama değeri	$\frac{1}{n}\left(\sum_{i=1}^{n} c_i\right)$	c, Java Sanal Makinesi'nde en son elde edilen işlemci yüküdür.
M	Hafıza kullanım oranının ortalama değeri	$\frac{1}{n}\left(\sum_{i=1}^{n} H_i\right)$	H hafıza kullanım oranıdır.
X	Saniyede gelen kayıt sayısının ortalama değeri	$\frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)$	x saniyelik birim zamanda, seçilen operatör veya göreve ulaşan kayıt sayısıdır.
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	$\frac{Y}{X}$	-
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 dizi
W	Lambda eşik sabiti	-	Dışarıdan parametre olarak verilir, sistemin ölçeklenmeye geçeceği limiti gösterir. Sabit bir değerdir.
λ	Lambda	$\frac{1}{n}\left(\sum_{i=1}^{n} J_i\right)$	Lambda değerlerinin ortalama değeri

Dinamik Ölçekleme Sistemi kısmında ise *metrik toplayıcı*, *performans ölçüm modeli* ve *sistem ölçekleme* olmak üzere üç temel modül yer almaktadır. *Metrik toplayıcı modülü*, Apache Flink kümesinin metriklerini belirli aralıklarla alır ve Kafka kümesine atar. Ölçekleme hesabının yapılacağı verilerin tamamı burada oluşur. Belirlenen bu kritik metrikler Kafka kümesine gönderildikten sonra bir Apache Flink kümesi bu metrikleri gerçek zamanlı olarak dinler ve veri parçaları üzerinde çeşitli hesaplamalar yapar. Bu hesaplamalar *performans ölçüm modelinde* gerçekleşir ve hesaplamaların sonucuna göre sistemin ölçeklenmesi gerekip gerekmediğine karar verilir. Sistemin ölçeklenmesinin gerektiği durumda *sistem ölçekleme modülü*, 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.

Önerilen modelin özelliklerinden bir tanesi de gecikmeye duyarlı bir ölçekleme sistemi sunmasıdır. Akan veri işleyen sistemlerde veri örneklerinin kuyrukta bekleme süreleri ile servis/işleme süreleri gecikmelere yol açmaktadır. Sistem, bu gecikmelerinde göz önüne alınarak veri akış trafiğinin değerlendirilmesi ile işlem hacmi azaltılacak ya da arttırılacak şekilde ölçeklendirilebilir. Kuyruk Teorisi (Botran, Alonso ve Lozano, 2014), sisteme gelen işler ile sistemden ayrılan işler arasındaki ilişkiyi modellemek için Internet uygulamaları ve sunucu modellemede yaygın olarak kullanılmaktadır. Bu teori, kuyruk uzunluğu veya isteklerin ortalama bekleme süresi gibi performans ölçümlerini tahmin etmek gibi görevler için elastik uygulamalarda dinamik ölçeklendirme için de kullanılabilir. Kuyruk sisteminin performansını değerlendirmek için kullanılan en temel ölçümlerden bir tanesi trafik yoğunluğudur. Bu çalışmada da önerilen modelde trafik yoğunluğunu hesaplamak için Kuyruk Teorisi'nden yararlanılmıştır. Model içerisinde her bileşen bir kuyruk sistemi olarak düşünülebilir. t değişkeni, ardışık iki gelen veri örneği arasındaki zaman aralığını ve u ise gelen bir veri örneğinin servis/işleme süresini göstermek üzere trafik yoğunluğu (α) aşağıdaki şekilde hesaplanabilir :

$$\alpha = \frac{E(u)}{E(t)} \tag{1}$$

t değişkeni rastgele olarak seçilirken, u değişkeninin tüm varış zamanları için bağımsız ve aynı şekilde dağılmış olduğu kabul edilir. Denklem 1'de $E(u)$ ortalama servis süresini, $E(t)$ ise ortalama varışlar arası geçen süreyi göstermektedir. Buna göre, trafiğin sunuculara eşit olarak dağıtıldığı varsayıldığında z adet sunucu için, sunucu kullanımı ve her sunucunun meşgul olduğu sürenin ortalama oranı denklem 2 ile gösterilebilir:

$$\delta = \frac{\alpha}{z} = \frac{E(u)}{zE(t)} \tag{2}$$

Dolayısıyla, α trafik yoğunluğu, gerekli sunucu sayısı (z) ve tıkanıklığın (δ) bir ölçüsüdür. Modelde $\delta \to 1$ ve $\alpha \to z$ alınarak sunucu sayısı tıkanıklığı ortadan kaldırmak için gereken örnek sayısına çevrilir. Her örnek tek sunuculu G/G/1 kuyruk sistemi olarak ele alınmıştır. Bu kuyruk sisteminde G, hem varış süreleri hem de servis/işleme sürelerindeki genel dağılımı temsil etmektedir. 1 ise modelin tek sunuculu bir sistem olduğunu gösterir. Farklı varış ve işleme süreleri birbirinden bağımsız olarak düşünülür. Buna göre, $\gamma = \frac{1}{E(t)}$ ortalama varış oranını, $\mu = \frac{1}{E(u)}$ servis oranını ve $x \in \{t, u\}$ olmak üzere $z_x = \frac{\sqrt{Var(x)}}{E(x)}$ varyasyon katsayısını göstermektedir. $\delta = \alpha = \gamma/\mu$ ve $\delta \to 1$ olması durumunda yüksek veri akışının olduğu yoğun trafik altındaki bir kuyruk sistemi için, Kingman Formülü (Kingman, 1962) kullanılarak, n veri örneği sayısını göstermek üzere, bir örneğin kuyrukta harcaması beklenen ortalama süre denklem 3 ile hesaplanabilir:

$$T_n = \left(\frac{\delta}{1-\delta}\right)\left(\frac{z_t^2 + z_u^2}{2}\right)E(u) \tag{3}$$

Sonuç olarak, ρ mevcut veri örneği çıktı süresini göstermek üzere sistemde bir örnek için meydana gelmesi beklenen gecikme denklem 4 ile hesaplanır:

$$\tau_n = T_n + (u - p) \tag{4}$$

Sistemin toplam çalışma süresi sistem yükünü göstermektedir ve kuyruk süresi de zamanla çalışma süresine yakınsama eğilimindedir. Kuyruk süresinin çalışma süresinden daha fazla olduğu durumlar veri trafiğinin düşük olduğu şeklinde yorumlanabilir. Bu tip senaryolarda, kuyruk süresi çalışma süresine yaklaşana dek işlem hacmi arttırılacak şekilde sistem ölçeklenebilir. Aksi durumda işlem hacmi azaltılarak sistemde meydana gelebilecek tıkanıklıklar önlenebilir. Geliştirdiğimiz sistemin ölçeklenmesine karar vermek için Kuyruk Teorisi ile birlikte çeşitli matematiksel ve

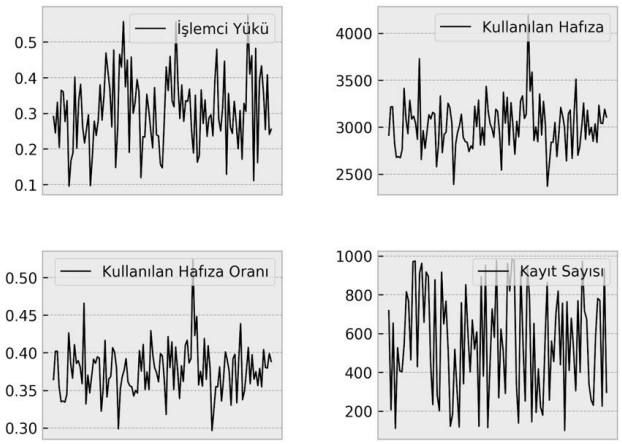
istatiksel hesaplamalar da yapılmıştır. Bu çalışmada kullanılan sistem metrikleri ve sabitler ile bunlara ait hesaplamalar Tablo 1'de açıklanmıştır.

Tüm bu hesaplamaların sonucunda gecikmeye duyarlı λ değeri performans ölçüm modeli içerisinde değerlendirilir. λ değeri, W sabitinden büyükse ölçekleme işlemi tetiklenir. Önerilen bu dinamik model, halihazırda çalışan uygulamalarda sistem iş yükünün ölçeklendirilmesi için kullanılmasının yanı sıra iş yükünü tahmin ederek ölçeklendirme işlemini tetikleyen modellere de entegre edilebilir. Modelin ön plana çıkan diğer önemli bir özelliği de sistemlerin servis/işleme oranlarını kontrollü bir performans ve kalite kaybıyla önemli ölçüde artırabilmesidir.

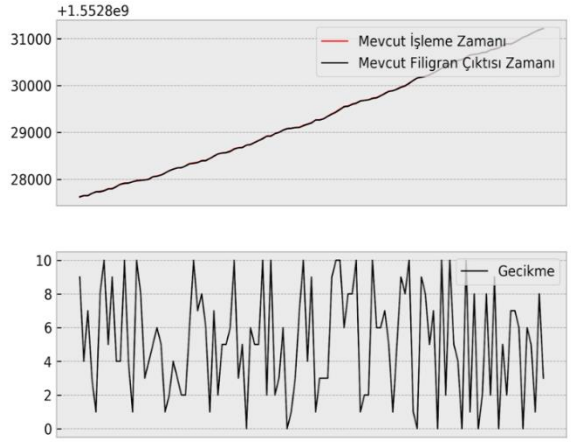
3. ANALİZ

Bu bölümde, önerilen gecikmeye duyarlı ölçekleme modelinin etkinliği, Apache Flink platformu üzerinde Scala programlama dili kullanılarak gerçekleştirilen çeşitli benzetim çalışmaları ile gösterilmiştir. Yapılan benzetim çalışmalarında, Q sabiti 30, W sabiti 0,6 olarak alınmıştır. Sabit değerler keyfi belirlenmiştir. Q sabiti gecikmelere karşı sistemin duyarlılığını saniye cinsinden, W sabiti sistemin tüm eşik dengesinin ne kadar olacağına göre belirlenebilir. W sabiti, 0 ile 1 arasında bir değer kabul eder. Benzetim çalışmasında kullanılan kritik metrikler işlemci yükü, hafıza kullanımı, gelen kayıt sayısı, mevcut işleme zamanı, mevcut çıktı filigranı ve gecikme değerleri olarak belirlenmiştir. Örnek veri setleri olasılık dağılımları kullanılarak sentetik bir yük üretilerek oluşturulmuştur.

Şekil 2a'da verinin geliş sırasıyla birinci pencerede gelen işlemci yükü, kullanılan hafıza, kullanılan hafızanın toplam hafızaya oranı ve saniyede gelen kayıt sayısı metrikleri görülmektedir. Şekil 2b sistemin gelen veriyi işleme zamanı ile filigran oluşturduğu zamanı göstermektedir. Daha önce de bahsedildiği gibi ikisi arasındaki fark ve kuyrukta bekleme süresi gecikmeyi vermektedir ve bu veri de yine Şekilde 2b'de gösterilmektedir. Şekil 2a ve Şekil 2b çizelgelerini oluşturan ve sisteme giriş olarak verilen girdilerin örnek bir kesiti ise Tablo 2'de verilmiştir. Çizelgelerde ve tablolarda bulunan veriler giriş verilerini temsil etmektedir. Tablolarda verinin kesiti gösterilmektedir.



Şekil 2a. Birinci iterasyon için işlemci yükü, kullanılan hafıza, kullanılan hafıza oranı ve kayıt sayısı



Şekil 2b. Birinci iterasyon için mevcut işleme zamanı, mevcut filigran çıktısı zamanı ve gecikme değerleri

Tablo 2'de gösterilen veri örnekleri ve metrikler için λ değeri W sabitinden küçük olduğu için ölçeklemeye ihtiyaç yoktur.

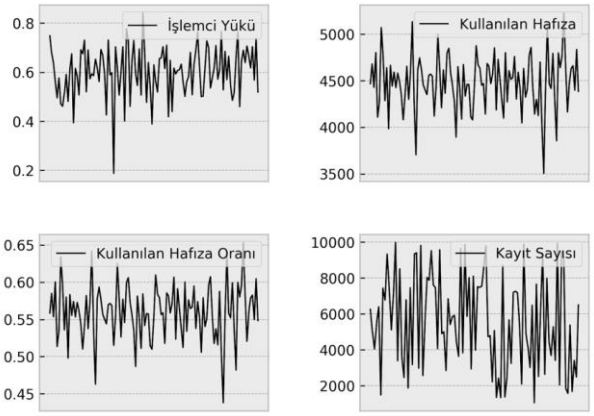
Bir önceki iterasyonda olduğu gibi Şekil 3a'da verinin geliş sırasıyla ikinci pencerede gelen işlemci yükü, kullanılan hafıza, kullanılan hafızanın toplam hafızaya oranı ve saniyede gelen kayıt sayısı metrikleri görülmektedir. Şekil 3b sistemin gelen veriyi işleme zamanı ile filigran oluşturduğu zamanı ve gecikmeyi göstermektedir. Şekil 3a ve Şekil 3b çizelgelerini oluşturan ve sisteme giriş olarak verilen girdilerin örnek bir kesiti ise Tablo 3'te verilmiştir. Buradan birim zamanda gelen kayıt sayısının artışına bağlı olarak, sistemin işlemci yükü ve hafıza kullanımının gözle görülür bir şekilde arttığı görülmektedir. Fakat Tablo 3'de görüleceği üzere λ değeri halen W sabitinden küçük olduğu için ölçeklemeye ihtiyaç yoktur; dolayısıyla ölçekleme tetiklenmez.

Tablo 2. Birinci hesaplama iterasyonu için veri örnekleri

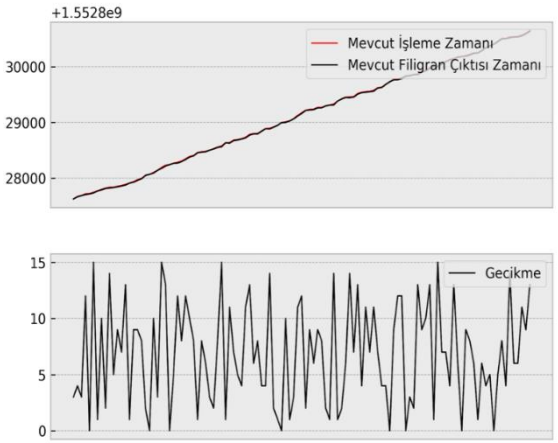
Metrik İsmi	Değer
İşlemci yükü	[0,29081359 0,24484373 0,33099842 0,20422209 0,36440966 0,36044093]
Kullanılan hafıza	[2915,05236578 3215,18824778 3217,2999415 2826,96742778]
Kalan hafıza oranı	[0,36438155 0,40189853 0,40216249 0,35337093 0,33482182 0,33580286]
Saniyede kalan gelen kayıt sayısı	[718, 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]
λ	**0,2926817375716746**

Tablo 2'de gösterilen veri örnekleri ve metrikler için λ değeri W sabitinden küçük olduğu için ölçeklemeye ihtiyaç yoktur.

Bir önceki iterasyonda olduğu gibi Şekil 3a'da verinin geliş sırasıyla ikinci pencerede gelen işlemci yükü, kullanılan hafıza, kullanılan hafızanın toplam hafızaya oranı ve saniyede gelen kayıt sayısı metrikleri görülmektedir. Şekil 3b sistemin gelen veriyi işleme zamanı ile filigran oluşturduğu zamanı ve gecikmeyi göstermektedir. Şekil 3a ve Şekil 3b çizelgelerini oluşturan ve sisteme giriş olarak verilen girdilerin örnek bir kesiti ise Tablo 3'te verilmiştir. Buradan birim zamanda gelen kayıt sayısının artışına bağlı olarak, sistemin işlemci yükü ve hafıza kullanımının gözle görülür bir şekilde arttığı görülmektedir. Fakat Tablo 3'de görüleceği üzere λ değeri halen W sabitinden küçük olduğu için ölçeklemeye ihtiyaç yoktur; dolayısıyla ölçekleme tetiklenmez.



Şekil 3a. İkinci iterasyon için işlemci yükü, kullanılan hafıza, kullanılan hafıza oranı ve kayıt sayısı



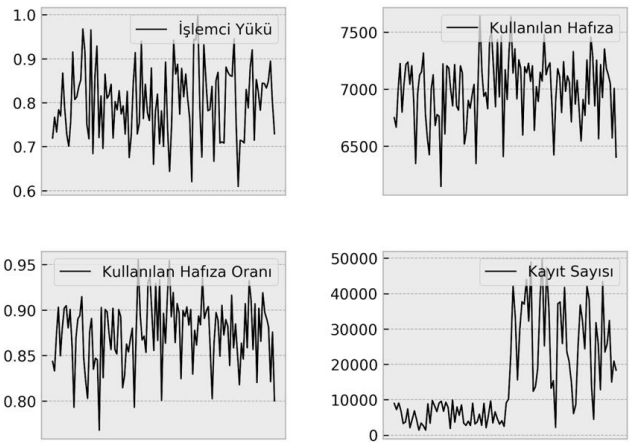
Şekil 3b. İkinci iterasyon için mevcut işleme zamanı, mevcut filigran çıktısı zamanı ve gecikme değerleri

Tablo 3. İkinci hesaplama iterasyon için veri örnekleri

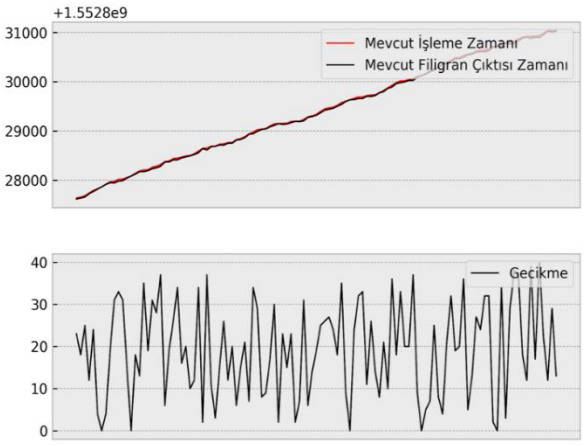
Metrik İsmi	Değer
İşlemci yükü	[0,74792182 0,67820917 0,63737983 0,56141087 0,49553547 0,57619555]
Kullanılan hafıza	[4467,48624105 4679,77961699 4427,77417013 4802,37689798]
Kalan hafıza oranı	[0,55843578 0,58497245 0,55347177 0,60029711 0,51352895 0,53401568]
Saniyede kalan gelen kayıt sayısı	[6248, 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 iterasyona benzer şekilde Şekil 4a'da verinin geliş sırasıyla üçüncü pencerede gelen işlemci yükü, kullanılan hafıza, kullanılan hafızanın toplam hafızaya oranı ve saniyede gelen kayıt sayısı metrikleri görülmektedir. Şekil 4b'de sistemin gelen veriyi işleme zamanı ile filigran oluşturduğu zamanı ve gecikmeyi göstermektedir. Şekil 4a ve Şekilde 4b çizelgelerini oluşturan ve sisteme giriş olarak verilen girdilerin örnek bir kesiti ise Tablo 4'te verilmiştir. Birim zamanda gelen kayıt sayısının anormal bir şekilde arttığı ve tepe yaptığı görülmektedir. 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'te görüleceği üzere λ değeri bu iterasyonda W sabitinden büyük olduğu için sisteme ölçekleme isteği gönderilir.

Sistem ölçeklendikten sonra sistemin durumu, işlemci yükü, kullanılan hafıza, kullanılan hafıza oranı ve birim zamanda gelen kayıt sayısı Şekil 5a'daki gibidir. Bu şekilden görüldüğü gibi birim zamanda gelen kayıt sayısı aynı davranışı göstermektedir. 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 5b'de görüldüğü gibi gecikmeler de kabul edilebilir bir noktaya gelmiştir. Şekil 5a ve Şekil 5b çizelgelerini oluşturan ve sistemin mevcut durumdaki metriklerinin örnek bir kesiti ise Tablo 5'de verilmiştir.



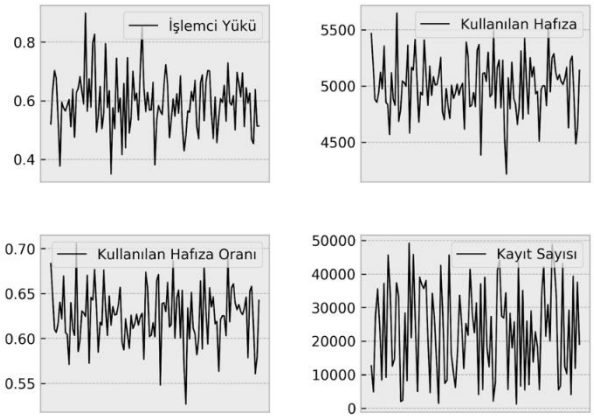
Şekil 4a. Üçüncü iterasyon için işlemci yükü, kullanılan hafıza, kullanılan hafıza oranı ve kayıt sayısı



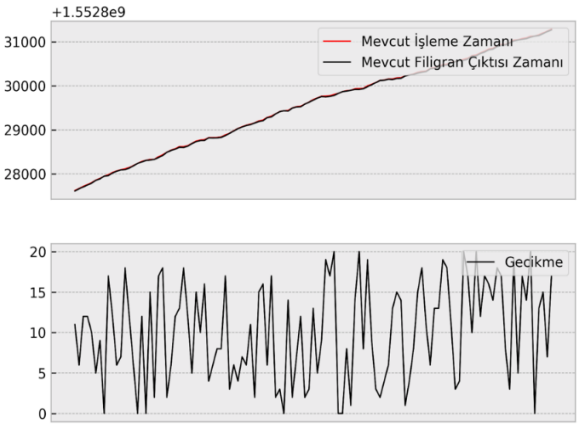
Şekil 4b. Üçüncü iterasyon için mevcut işleme zamanı, mevcut filigran çıktısı zamanı ve gecikme değerleri

Tablo 4. Üçüncü hesaplama iterasyon için veri örnekleri

Metrik İsmi	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**



Şekil 5a. Ölçeklemeden sonra işlemci yükü, kullanılan hafıza, kullanılan hafıza oranı ve kayıt sayısı



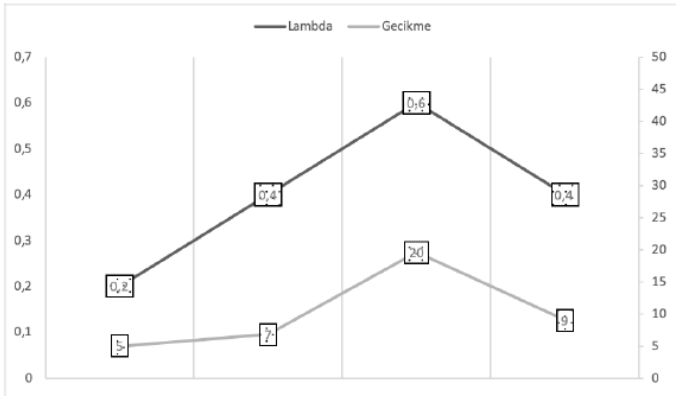
Şekil 5b. Ölçeklemeden sonra mevcut işleme zamanı, mevcut filigran çıktısı zamanı ve gecikme değerleri

Tablo 5. Sistem ölçeklendikten sonra veri örnekleri

Metrik İsmi	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**

Böylece, sistem için ölçekleme isteği gönderildikten sonra Apache Flink kümesi gecikmeye duyarlı λ değerinin küçülmesi ile birlikte daha etkili bir şekilde çalışır hale gelmiş ve sistemde meydana gelebilmesi muhtemel bir tıkanıklığın önüne geçilmiştir.

Şekil 6'da λ değeri ile gecikme arasındaki ilişki gösterilmiştir. Buradan 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.



Şekil 6. Lambda ve gecikme arasındaki ilişki grafiği

4. SONUÇLAR VE TARTIŞMALAR

Kurumlar müşterileri, uygulamaları ve ürünleri daha iyi tanımak ve anlamak için 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 sağlamaktadır. Fakat gerçek zamanlı akan veriyi işleyen sistemleri inşa etmek ve operasyonunu sağlamak oldukça zordur. Maliyeti de göz önünde bulundurarak sistemin gerçek zamanlı bir şekilde çalışmaya devam edebilmesi için dağıtık sunucuların yönetilebilmesi ve gerektiğinde kolayca ölçeklenebilmesi gerekmektedir.

Bu çalışmada akan veriyi işleyen dağıtık sistemlerin, değişen iş yüklerine adapte olması adına ölçeklenebilirlik problemlerine odaklanarak, Apache Flink üzerinde gecikmeye duyarlı ölçeklendirme

yapabilen bir model önerilmektedir. Önerilen model ile tüketen tarafın kaynakları ve çalışan iş özelindeki metrikleri kullanılarak sistemin o an ölçeklenmesi gerekip gerekmediğine karar verilmesine olanak sağlanmıştır. Bunun yanında, ölçekleme sürecinin daha soyut bir şekilde yapılması ve farklı akan veriyi işleyen dağıtık sistemlere de kolay entegre olabilmesi amaçlanmıştır. Ayrıca, model içerisinde trafik yoğunluğu ve kuyrukta bekleme süreleri, Kuyruk Teorisi ve Kingman Formülü ile hesaplanarak sistemde meydana gelmesi beklenen gecikmelere göre tıkanıklıkların önlenmesi de sağlanmıştır. Sonuç olarak, akan veriyi işleyen dağıtık sistemlerde kullanılabilecek bir sistem önerisinde bulunularak, bu sistemin etkin bir şekilde çalıştığı çeşitli benzetim çalışmaları ile desteklenerek gösterilmiştir.

Gelecek çalışmalarda hesap yapılan metrik verilerini özelleştirip türeterek metrik sayısını arttırmak, tüketen tarafın metriklerini de hesaba katmak ve metrik verilerinin belirli bir matematiksel ve istatistiksel hesap yerine, makine öğrenmesi algoritmaları kullanılarak daha etkin bir ölçekleme stratejisi oluşturmak gibi planlamalar mevcuttur. Bundan sonraki aşamada, mevcut çalışmaya planlar dahilinde belirlenen yeni özellikler sırasıyla eklenerek sistemin geliştirilmesi üzerine çalışılacaktır.

KAYNAKLAR

Basanta-Val. P., Garcia N., Fernandez L., Fiesteus J. 2017. Patterns for Distributed Real-Time Stream Processing, IEEE Transactions on Parallel and Distributed Systems, 28(11), 3243-3257.

Botran T.L., Alonso J.M., Lozano J.A. 2014. A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments, Journal of Grid Computing, 12(4), 559-592.

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, 28(12), 3553-3569.

Chakraborty R., Majumdar S. 2016. A Priority Based Resource Scheduling Technique for Multitenant Storm Clusters, International Symposium on Performance Evaluation of Computer and Telecommunication Systems, pp1-6, 24-27 Temmuz, Kanada.

Chen H., Zhang F., Jin H. 2017. Popularity-aware Differentiated Distributed Stream Processing on Skewed Streams, IEEE 25th International Conference on Network Protocols, pp1-10, 10-13 Ekim, Kanada.

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, pp61-68, 6-8 Mart, Rusya.

Farahabady M.R.H., Samani H.R.D., Wang Y., Zomaya A.Y., Tari Z. 2016. A QoS-Aware Controller for Apache Storm, IEEE 15th International Symposium on Network Computing and Applications, pp334-342, 31 Ekim-2 Kasım, ABD.

HoseinyFarahabady M., Zomaya A.Y., Tari Z. 2017. QoS- and Contention- Aware Resource Provisioning in a Stream Processing Engine, IEEE International Conference on Cluster Computing, pp137-146, 5-8 Eylül, ABD.

Khoshkbarforoushha A., Ranjan R., Gaire R., Abbasnejad E., Wang L. 2016. Zomaya A.Y., Distribution Based Workload Modelling of Continuous Queries in Clouds, IEEE Transactions on Emerging Topics in Computing, 5(1), 120-133.

Kingman J.F.C. 1962. On queues in heavy traffic, Journal of the Royal Statistical Society. Series B (Methodological), 24(2), 383-392.

Kombi R.K., Lumineau N., Lamarre P. 2017. A preventive auto-parallelization approach for elastic stream processing, IEEE 37th International Conference on Distributed Computing Systems, pp1532-1542, 5-8 Haziran, ABD.

Li T., Tang J., Xu J. 2016. Performance Modeling and Predictive Scheduling for Distributed Stream Data Processing, IEEE Transactions on Big Data, 2(4), 353-364.

Liu X., Buyya R. 2017. Performance-Oriented Deployment of Streaming Applications on Cloud, IEEE Transactions on Big Data, 5(1), 46-59.

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, pp259-266, 27 Haziran-2 Temmuz, ABD.

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, pp623-630, 12-14 Aralık, Avustralya.

Renart E.G., Diaz-Montes J., Parashar M. 2017. Data-driven Stream Processing at the Edge, IEEE 1st International Conference on Fog and Edge Computing, pp31-40, 14-15 Mayıs, İspanya.

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, pp848-857, 14-17 Mayıs, İspanya.

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, 29(12), 2669 – 2681.

Wang Y., Tari Z., HoseinyFarahabady M., Zomaya A.Y. 2017. QoS-aware resource allocation for stream processing engines using priority channels, IEEE 16th International Symposium on Network Computing and Applications, pp1-9, 30 Ekim-1 Kasım, ABD.

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, pp22-31, 4-8 Nisan, Almanya.

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, pp372-279, 12-14 Aralık, Avustralya.