IOT İÇİN MESSAGE BROKER TASARIMI VE MOCK DATA İLE VERİ İŞLEME SÜRELERİNİ KARŞILAŞTIRMA

Süha UĞUZ TURGUT

Cumhuriyet Üniversitesi

20239257005@cumhuriyet.edu.tr - 0009-0006-0660-5918

Ahmet Turan KARAKUŞ

Cumhuriyet Üniversitesi

20239257014@cumhuriyet.edu.tr - 0009-0001-2941-3250

Ahmet Gürkan YÜKSEK

Cumhuriyet Üniversitesi

agyuksek@cumhuriyet.edu.tr – 000-001-7709-6360

ÖZET

Internet of Things (IoT) yani bilinen adıyla Nesnelerin İnterneti derin bir teknoloji çerçevesi olmakla birlikte küreselleşen bir alandır. Bu teknoloji ile kullanılan cihazlar farklı yöntemlerle veri alışverişi sağlar. Nesnelerin İnterneti ekosistemi, her geçen dakika artan veri hacmiyle başa çıkmak için etkin veri yönetimi ve güçlü iletişim stratejilerine ihtiyaç duymaktadır. Özellikle makine öğrenimi ve gömülü teknolojilerden yararlanarak anlık veya düzenli veri toplayan IoT cihazlarının günden güne sayısının artması ve bu toplanan verilerin işlenme sürelerinin yeni gelen verilerle orantılı olarak giderek artması yeni sistem tasarımlarınn geliştirilmesine yol açmıştır. Bu devasa büyüklükteki verilerin toplanması ve işlenmesi konuları hala üzerine çalışmalar yapılan, iyileştirmeler sunulan bir konudur. Bu veri yoğunluğunu kontrol altına alabilmek için örnek bir çözüm, Message Broker tasarımlarıdır. Özellikle anlık veri işlemesine ihtiyaç duyan görme engelli bireylere yardımcı olma amacı taşıyan yapay zekâ destekli gömülü sistemler, araç otonom sistemleri, konum takibine dayalı anormal durum tespit sistemleri gibi gerçek zamanlı sistemlerde, cihazlar arası iletişimi koordine etmek ve veri akışını optimize etmek oldukça önemli hale gelmiştir. Devasa boyuttaki bu IoT verilerinin, koordine bir şekilde performanslı olarak işlenebilmesi sürecinde, standart haline gelen İlişkisel Veritabanı

Sistemleri (RDMS), Standart Mesaj Kuyruklama sistemleri (RabbitMQ, Apache Kafka gibi) de bazen yetersiz kalmaktadır. Bu çalışma, farklı broker tasarımlarını ele almayı ve Mock Data kullanılarak yapılan işleme sürelerini NoSQL veritabanı üzerinde analiz etmeyi amaçlamaktadır. Ayrıca, veri sanallaştırma teknolojisi kullanarak çoğaltılmış Message Broker tasarımlarının performans metriklerini incelemektedir. Sonuçlar, Grafana aracılığıyla analitik ve etkileşimli bir şekilde görselleştirilerek sunulmaktadır. Böylece, farklı broker tasarımlarının anlamlı bir şekilde değerlendirmesi yapılmaktadır. Aynı zamanda farklı broker tasarımları arasındaki performans farklarını hafıza, bant genişliği, işlemci kullanımı gibi görselleştirilmiş metrikler ile de ölçmek hedeflenmektedir.

Anahtar Kelimeler: IoT, Message Broker, Mock Data, Veri İşleme, Veri Karşılaştırma

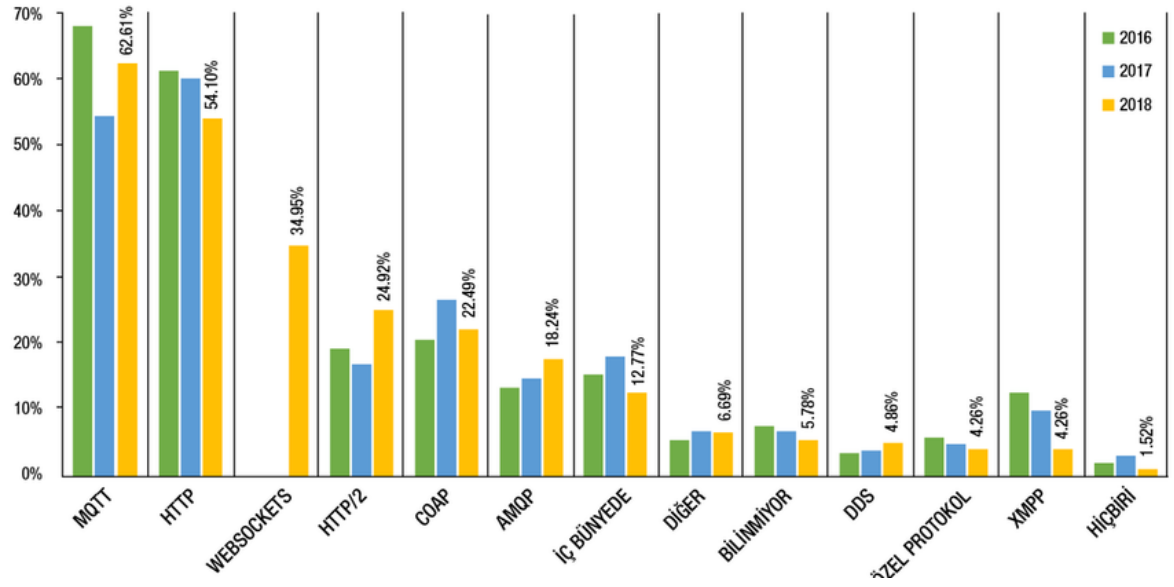
1.NESNELERİN İNTERNETİ ve BROKER SİSTEMLERİ

Nesnelerin interneti, uzun yıllardır bilinen bir kavram olsa da son yıllarda yapay zekâ ile tekrar en hızlı gelişen teknolojilerden biri olarak yükselişe geçmiştir. Nesnelerin İnterneti hem cihazlar ile bulut arasındaki hem de cihazların kendi aralarındaki iletişimi mümkün kılan teknolojilerin tamamını ifade eder [1]. Bu teknoloji, endüstriyel sistemlerden ev otomasyonuna, sağlık sektöründen tarıma kadar geniş bir yelpazede uygulanmaktadır. Nesnelerin interneti, farklı iletişim protokolleri ile birbirleriyle haberleşebilen, algılama ve veri işleme kabiliyetine sahip nesnelerden oluşan küresel bir ağdır [2]. Farklı protokollerin özellikleri ve kullanım alanları, tasarlanan sisteme göre tercih edilme durumlarını belirlemektedir.

IoT sistemlerinde veri iletişimi ve yönetimi önemli bir konudur. Bu noktada broker sistemleri devreye girer. Broker sistemleri, IoT cihazları arasında veri alışverişini kolaylaştıran aracı yazılımlardır. Bunlar, veri akışını düzenler, güvenliği sağlar ve veri işleme süreçlerini yönetir. Örneğin, bir IoT cihazı sensörlerden gelen verileri bir broker aracılığıyla bir hedef cihaza iletebilir.

Mevcut broker sistemlerinden bazıları; MQTT (Message Queuing Telemetry Transport), AMQP (Advanced Message Queuing Protocol) ve CoAP (Constrained Application Protocol), RabbitMQ, Apache Kafka ve ActiveMQ gibi protokollere dayanan çözümlerdir. Bu sistemler farklı ihtiyaçlara yönelik tasarlanmış olup genellikle düşük güç tüketimi, düşük bant genişliği kullanımı ve yüksek ölçeklenebilirlik gibi özellikler sunarlar.

Bu çalışmada, IoT sistemlerinde broker sistemlerinin önemi ve yaygın kullanım alanları incelenecek ve mevcut broker sistemlerinden örnekler verilecektir. Bu analiz, IoT teknolojilerinin yaygın benimsenmesinde ve günlük hayatta daha fazla entegrasyon sağlanmasında önemli bir rol oynamaktadır.



Görsel 1. IoT Uygulamalarında Tercih Edilen Protokoller [3]

1.1 BROKER SISTEMLERINDEKİ PROTOKOLLERİN AYRINTILI İNCELENMESİ

1.1.1. MQTT Protokolü (Message Queuing Telemetry Transport)

Standart tabanlı bir mesajlaşma protokolü veya kurallar dizisidir. Cihazdan cihaza iletişim için kullanılır. Protokolün uygulanması basittir. IoT verisini kolayca iletebildiğinden, IoT cihazları bu tarz veri iletimi için MQTT'yi kullanır [4].

1.1.2. HTTP Protokolü (Hyper-Text Transfer Protocol)

Dünyada en yaygın kullanılan protokoldür. HTTP'nin en bilindik kullanım alanı internetteki web sayfalarını görüntülemek için kullanılan protokol olmasıdır [5].

1.1.3. Web Sockets Protokolü

Bir istemci ve bir sunucu -ki bunlar genelde tarayıcı ve sunucu olmaktadır- arasında bir bağlantı oluşturmak ve aralarında gerçek zamanlı olarak iletişimi sağlamak için geliştirilmiş bir protokoldür [6].

1.1.4. HTTP/2 Protokolü (Web 2.0)

HTTP/2 protokolü erişim sağlanan cihaz ve sunucu arasındaki bilgi alışverişinin sağlanmasıyla ilgili kurallar ve yöntemleri düzenleyen bir protokoldür [7]. Web 2.0 olarak da bilinir.

1.1.5. CoAP Protokolü (Constrained Application Protocol)

IoT cihazları arasında düşük güç tüketimi ve düşük kaynak kullanımı ile haberleşmeyi sağlamak amacıyla tasarlanmış bir uygulama iletişim katmanı protokolüdür [8].

1.1.6. AMQP Protokolü (Advanced Message Queuing Protocol)

Mesajlaşma uygulamaları arasında güvenli ve etkili bir şekilde veri alışverişi yapmak için tasarlanmıştır. Açık kaynaklı bir mesajlaşma protokolüdür [9].

1.1.7. DDS Protokolü (Data Distribution Service)

Geniş ölçekli dağıtık sistemler geliştirmeyi kolaylaştırmak için Object Management Group (OMG) tarafından yayınlanmıştır. Yayınla/abone ol mimarisini destekleyen veri odaklı bir ara katman yazılımıdır [10].

1.1.8 XMPP Protokolü (Extensible Messaging and Presence Protocol)

Daha önceki adıyla Jabber, Internet'teki iki ucun herhangi bir yapısal bilgiyi birbirleri arasında karşılıklı ve neredeyse eş zamanlı aktarmalara olanak sağlayan açık kaynaklı bir protokoldür [11].

2. MESSAGE BROKER TASARIMI

Broker sistemlerinin temel bileşenleri; veri akışını yönetmek, güvenlik sağlamak ve sistem genelinde yüksek performans bunun yanı sıra da verimlilik sunmak üzere tasarlanmıştır. Çalışmamızda; RabbitMQ, Apache ActiveMQ ve Apache Kafka teknolojileri kullanarak Java kodlama dili üzerinde geliştirilen broker sistemi işlenecektir. IoT cihazlarından gelen verilerin toplanması, işlenmesi ve dağıtılması işlevlerini gerçekleştirmek için geliştirilmiştir.

Brokerlar amaçları ve kullanım alanları açısından farklı özelliklere sahiptir. Apache ActiveMQ, Java Mesaj Servisi'ni (JMS) benimser. Mesajlaşma ve güvenlik konusunda yüksek performans verir. Apache Kafka, yüksek hacimli veri akışlarını yönetmek için kullanılır. Dayanıklılığı ve hızıyla ön plana çıkan Apache Kafka'nın yanı sıra RabbitMQ ise özellikle karmaşık mesajlaşma ihtiyaçları için tercih edilir. Esnek ve güvenilir bir aracı yazılımdır. Geliştirilen broker tasarımında, veri akışını düzenleyen protokoller ve algoritmalar, bu iki sistem –Apache ActiveMQ ve RabbitMQ- aracılığıyla uyumlu hale getirilmiştir. Güvenlik ve veri bütünlüğünü sağlamak için ise şifreleme, erişim kontrol listeleri ve mesaj doğrulama gibi önlemler alınmıştır. Bu tasarım, verilerin güvenliğini maksimum düzeyde tutarak IoT uygulamaları için kritik öneme sahip veri bütünlüğünü korumaktadır. Ayrıca geniş ölçekli IoT uygulamalarında broker sistemlerinin nasıl optimize edilebileceği ve güvenli bir şekilde yönetilebileceği konusunda değerli görüşler sunmaktadır.

2.1. MOCK DATA OLUŞTURMA

Mock data, yazılım geliştirme ve test süreçlerinde kritik bir rol oynar, zira bu data gerçek sistem verilerinin karmaşıklığını ve davranışlarını taklit etmek için kullanılır. Mock data oluşturmak, önceden tanımlanmış veri setlerini kullanarak gerçekleştirilebileceği gibi tamamen rasgele bir şekilde de oluşturulabilir. Rastgele veri üretimi, daha geniş çeşitlilikte veri senaryoları sağlayarak sistemlerin sınırlarını zorlamayı amaçlar.

Gerçek veriye benzerlik, yapısal veya davranışsal benzerlikler olabilir. Yapısal benzerlik, mock verilerin gerçek veri setlerinin sahip olduğu özellikler (format, türler ve ilişkisel bağlar gibi) yapısal unsurları taklit etmesini içerir. Davranışsal benzerlik ise verilerin zamana bağlı değişimleri veya belirli kullanıcı etkileşimleri sonucu ortaya çıkan davranışları modellemeyi hedefler. Bu yaklaşımlar, test edilen sistemlerin gerçek dünya koşullarına ne kadar dayanıklı olduğunu anlamada yardımcı olur.

Mock data oluşturma araçları ve teknolojileri ise geniş bir yelpazeye sahiptir. Örneğin, Python programlama dili için Faker kütüphanesi, geliştiricilere çeşitli lokalizasyon seçenekleri ile gerçekçi veri setleri oluşturma imkânı sunar. JavaScript için kullanılan JSON Server, REST API'ler için hızlı prototipleme yapılmasını sağlar. Aynı zamanda GenFu kütüphanesi ise C# dili için veri setleri oluşturmaya yarayan bir kütüphanedir. Ayrıca, Postman gibi API test ve geliştirme araçları da mock servisler oluşturarak API tasarım ve test süreçlerinde büyük kolaylıklar sağlar. Bu araçlar, IoT projelerindeki gibi ağ tabanlı sistemlerin test edilmesinde özellikle faydalıdır. Çünkü gerçeğe yakın veri akışı simülasyonları yapmak, sistemlerin gerçek koşullarda nasıl performans göstereceğini önceden görebilmeyi mümkün kılabilir. Bu detaylı simülasyonlar sayesinde, geliştiriciler sistem hatalarını önceden tespit edebilir ve uygulamalarını bu hatalara karşı güçlendirebilir.

Sistemimizde gerçek veriye benzerlik sağlama stratejisi olarak, özel bir yöntem veya teknoloji kullanımına gerek duyulmamıştır. Bunun yerine, şehirler sabit olarak belirlenmiş ve random olarak uzaklık ve yakınlık mesafeleri üreten bir mekanizma geliştirilmiştir. Bu mesafeler, JSON formatında veri dosyaları olarak oluşturularak, veri brokerler aracılığıyla işlenmiş ve sonrasında Cassandra veritabanında saklanmıştır. Bu yöntem, gerçek dünya verilerine benzer koşullar yaratılmasını sağlarken, sistemin karmaşıklığını ve maliyetlerini de minimize etmektedir. Bu basit ama etkili yaklaşım, test ve geliştirme süreçlerinde verimliliği artırmaktadır. Doğru bir simülasyon yapılmasına da olanak tanımaktadır.

2.2 KULLANILAN TEKNOLOJİLER VE BU TEKNOLOJİLERİN TANITIMI

Bu projede kullanılan temel teknolojiler arasında Kubernetes (K8s) bulunmaktadır. Kubernetes, ölçeklenebilir ve yönetilebilir bir şekilde konteyner tabanlı uygulamaları dağıtmak, yönetmek ve otomatikleştirmek için tasarlanmış bir platformdur. K8s, uygulamaları otomatik olarak dağıtabilir, hizalayabilir ve yönetebilir, ayrıca yük dengelemesi ve otomatik ölçeklendirme gibi özellikler sunar. Bu proje içinde, K8s kullanılarak uygulamaların konteyner ortamlarında yönetimi ve dağıtımı sağlanmaktadır.

RabbitMQ, Apache ActiveMQ ve Apache Kafka gibi mesajlaşma sistemleri, asenkron veri iletişimi için önemli rol oynamaktadır. RabbitMQ, AMQP (Advanced Message Queuing Protocol) protokolünü uygular. Çeşitli programlama dilleri ve platformlar arasında iletişimi kolaylaştırır. Apache ActiveMQ, açık kaynaklı bir mesajlaşma aracıdır ve geniş bir protokol desteği sunar, özellikle JMS (Java Message Service) üzerinde odaklanır. Apache Kafka ise dağıtık, yüksek performanslı ve dayanıklı bir akış platformudur. Kafka, büyük ölçekli veri

akışlarını işlemek için tasarlanmıştır. Gerçek zamanlı veri akışlarını işleyerek veri entegrasyonunu kolaylaştırır.

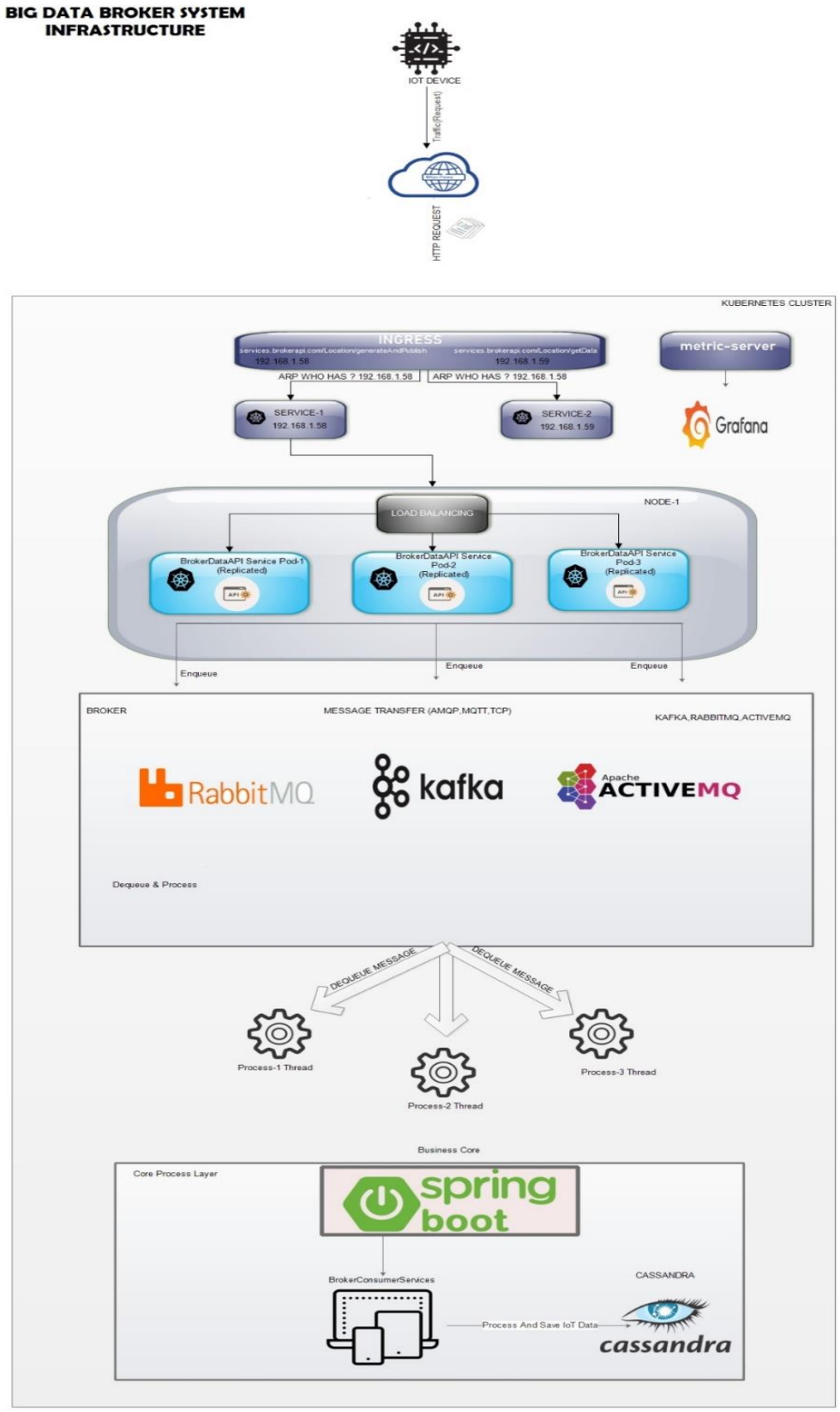
Apache Cassandra, yüksek ölçeklenebilir bir NoSQL veritabanıdır. Çok sayıda düşük gecikmeli yazma ve okuma işlemine olanak tanır. Ayrıca, dağıtık mimarisi sayesinde veri merkezleri arasında ve bulut ortamlarında kullanılabilir. Bu proje içinde, Cassandra veritabanı, büyük ölçekli veri akışlarını yönetmek ve depolamak için tercih edilmiştir.

Bu teknolojiler, uygulamanın güvenilirliğini, ölçeklenebilirliğini ve performansını artırmak için bir arada kullanılmaktadır. RabbitMQ [12], Kubernetes [13], Apache ActiveMQ [14], Apache Kafka [15] ve Apache Cassandra'nın [16] birlikte kullanımı, uygulamanın gereksinimlerini karşılamak için güçlü bir temel oluşturur. Karmaşık sistemlerin yönetimini ve dağıtımını kolaylaştırır.

Ayrıca, bu projede Grafana gibi güçlü bir veri görselleştirme aracı kullanılmaktadır. Grafana, çeşitli veri kaynaklarından gelen verileri alır ve bu verileri interaktif ve özelleştirilebilir görseller halinde sunar. Grafana'nın esnek ve kullanıcı dostu arayüzü, veri analizi ve izleme süreçlerini kolaylaştırır.

3. VERİ KAYBINA HASSASİYETLİ VE PERFORMANSINA ODAKLI ÖNERİLEN MESSAGE BROKER MİMARİSİ

Geniş kapsamlı veya büyük veri işleyen yazılım projelerinde Kubernetes (K8s), RabbitMQ, Apache ActiveMQ, Apache Kafka, Apache Cassandra gibi üçüncü parti uygulamaları kullanılarak tasarlanan sistemin; güvenli, dayanıklı, erişilebilir ve ölçeklenebilir olması sağlanmaktadır. Esasında bu kavramlara uyan bir biçimde, büyük veri ve performans odaklı sistemlerin oluşturulması, üçüncü parti yazılımlarının doğru altyapı tasarımına sahip olması ve kullanılacak amaca uygun araçların seçilmesine bağlıdır. Bu anlamda sanallaştırma ve açık kaynak kodlu projelerin kullanılması hem güncellemeye açık hem de diğer bulut sistemi sayesinde rahatlıkla diğer sistemler ile uyumlu çalışabilme olanağı sağlamaktadır [17]. Bu olanak ile günümüz dünyasında çoğalan IoT cihazlarının ve veri işleyen sistemlerin farklı konumlarda çalışması; farklı konumlardan hızlı ve güvenli olarak veriye ulaşabilmesi önemli bir problem hale gelmektedir. Bu problemin çözümünde veri merkezi tabanlı çalışabilen ve replikasyonlar ile büyük miktarda veriyi her zaman yönetebilir hale getiren Cassandra teknolojisi kullanılmıştır.



Görsel 2. Broker Sistem Mimarisi

ActiveMQ, RabbitMQ ve Kafka gibi broker sistemleri, farklı kullanım senaryolarına yönelik tasarlanmıştır. Örneğin, ActiveMQ genellikle geleneksel mesaj kuyruğu senaryolarında tercih edilirken, RabbitMQ esnek ve genel amaçlı bir mesaj broker olarak kullanılır. Apache Kafka ise, büyük veri akışlarını işlemek ve gerçek zamanlı veri işleme senaryolarında kullanılmak üzere tasarlanmıştır. Herhangi bir büyük veri senaryosu genellikle depolama ölçeklendirebilme ve veri hattının güvenliğine ilişkin endişeler taşımaktadır. Modern çözümler birden fazla alanda etkili olabilir. Ancak farklı broker sistemlerinin seçimi noktasında büyük veriyi otomatik hale getirebilmek için burada bahsedilen broker sistemleri kullanılan kaynaklar göz ardı edilmektedir. Ölçeklendirilmiş sistemlerde düzenli olarak yüksek performans sağlama gibi özelliklere sahip olsa da bant genişliği, hafıza, CPU ve IO kullanımları gibi kaynaklar göz ardı edilir. Bu noktada önerilen sistemde tam işlevsellik elde etmek için Kubernetes ve Docker bileşenlerinin yanı sıra NoSQL tabanlı bir veri depolama sistemi kullanılır. IoT ağındaki cihaz sayısının artmasına bağlı olarak veri hacmi artacağından, artan veri hacmini karşılayabilecek sistemin yönetilmesini ve yük dengelenmesinin sağlanması noktasında Kubernetes Orkestrasyon Arac tavsiye edilmektedir. Önerilen bu sistemin yönetilmesinde orkestrasyon yapmak üzere -işletim sistemlerinden bağımsız ve her sisteme kolay bir şekilde uyum sağlayabilmesi açısından- sanallaştırma altyapısında güçlü bir araç olan Docker teknolojisi tavsiye edilmektedir. Önerilen araçların birleştirilmesi ve kullanılması noktasında uygulama bazındaki ana bileşen, Spring Boot ile geliştirilmiş bir Java uygulaması üzerinde sistematik hale getirilmiştir. Bu uygulama, Spring Boot Web aracılığıyla IoT cihazlarından gelen verilerin Apache Kafka, RabbitMQ, Apache ActiveMQ gibi broker araçlarına yönlendirmesini ve bu verilerin saklanma noktasında da Cassandra veritabanını tavsiye eder. Yerel çalışma veya uzak sunucu çalışma ortamlarında kullanılmak üzere uygulamanın bağımlılıklarını yöneten Docker Compose dosyası ile uygulamanın, farklı servislerini ve bu servislerin nasıl birlikte çalışacağını belirtmek için kullanım sağlanmıştır. Docker teknolojisi ve Compose dosyası sayesinde, her bir servisin hangi portları kullanacağı, hangi ortam değişkenlerine sahip olacağı ve diğer servislere olan bağımlılıkları belirlenmiş; bu dosya Kubernetes teknolojisinde de kullanmak üzere referans alınmıştır. Böylelikle bu mimari; büyük veri işleyen ve yüksek kullanılabilirlik gerektiren uygulamalar için ideal bir ürün ortamını Görsel 2'de belirtildiği gibi sunmaktadır. Her bir servis, belirli bir görevi yerine getireceğinden ve birlikte çalışma sağlayabileceğinden Java tabanında yazılmış olan veri işleme sürecindeki sürdürülebilir-ölçeklendirilebilir uygulama işlevselliğini sağlamaktadır.

Görsel 2'de bulunan sistem mimarisinden hareketle şu çıkarımlar yapılabilir. Kubernetes, bir API (Application Programming Interface) aracılığıyla dinamik olarak belirlenen limitlerde veya Kernel tabanında ölçeklenen uygulamaları yürütebilen bir konteyner düzenleme sistemidir. Bu sistem, uygulamaların kullandığı kütüphaneleri işletim sisteminden soyutlayarak konteynerler içinde izole eder. Ayrıca, Kubernetes gerektiğinde bu mikro servis iş parçacıklarını ölçekleyebilir ve konumdan bağımsız uygulamaları dağıtabilir. Sistem, VLAN aracılığıyla farklı uygulamaların birlikte çalışmasını sağlar. Böylece uygulamaları yönetip düzene sokmuş olur. Kubernetes aslında konteynerları çalıştırmak için birlikte çalışan ana kontrolcü ve bağımlı çalışan düğümlerinden bir araya gelen kümedir. Bu kümelerden ana küme -master node-

bulunulan kümenin çalışanlarının (worker) ve kapsüllerin (POD) yönetildiği düğümdür. Çalıştırmak istediğimiz uygulamalar, çalışan düğüm olarak adlandırılan bu düğümde çalıştırılır.

Uygulamaları konteynerledikten sonra mikro servis bazında orkestra etmek için Kubernetes aracılığıyla master ve worker kümelerinde uygulamamızı hayata geçirebilir ve Kubectl Komut Satırı Aracı sayesinde bu küme ile iletişim kurup ilgili uygulamaları zaman ayarlı genişletme, yük altında kopya uygulama çalıştırma gibi işlevlerini gerçekleştirebilir hale gelir. Kubernetes, IoT gibi yüksek kapasitede çalışması gereken sistemlerin bakımını organize eden bir araçtır. Bu araç, gerektiğinde sistemleri otomatik olarak genişletebilir ve zaman ayarlı belirlenen emirleri yerine getirebilir. Ayrıca donanımların belirli özelliklerini özel uygulamalara ayırabilir. Kubernetes, herhangi bir felaket senaryosunda uygulamaları başka bir uzak ortamdaki sunucularda eşzamanlı olarak çalıştırıp, yönetebilir. Bu özellikler, sistem yöneticisi tarafından yönetilir. Uygulamaların ölçeklenebilirliğini ve genişletebilirliğini sağlar. Bu projede uygulamanın yürütme süreçlerini yönlendirmesi/yönetmesi için merkez bir beyin görevi görecek olan Kubernetes aracını, sistem mimarisinin iskeletine yerleştirilmesi önerilmektedir.

Önerilen sistem çerçevesinde, gerçek zamanlı veri akışının anlık olarak yükselme yaptığı veya sürekli olarak yoğun veri akışının olduğu IoT ortamlarında performanslı bir şekilde gelen verilerin işlenmesi, saklanması konusunda yardımcı olan bir sistem tasarımı ve bu sistem tasarımında önerilen broker sistemlerinin performans karşılaştırmasını sunmaktadır.

Önerilen sistemin çalışma prensibi IoT cihazlarından gelen veriler ile başlamaktadır, gelen veriler ilk olarak Java Spring Boot Web ile yazılan BrokerDataAPI Restful servis ile karşılanır. Bu veri istenilen broker tipine göre ayrıştırılan ve her bir broker tipi için özelleştirilen "Producer" adındaki sistem sayesinde kuyruğa aktarılır. Kuyruğa verilen özelleştirilmiş konfigürasyon durumlarına göre (RabbitMq, Apache ActiveMQ, Apache Kafka) "Consumer" adı verilen yapılar kuyruktaki veriyi okur. İstenilen işlemler gerçekleştirilip, Cassandra veri tabanında saklanır. Tasarlanan sistem Docker ile sanallaştırılarak, Kubernetes aracı ile birden fazla kopya oluşturma sayesinde paralel olarak veri işleyebilir hale getirilmiştir. Yine bu paralelliğin kontrolü ve yük dengesi dağılımı Kubernetes aracının "Ingress eklentisi" ile sağlanmıştır. Oluşturulan bu sistemin çalışma prensibi Görsel 2'de belirtilmiştir.

4. PERFORMANS KARŞILAŞTIRMALARI VE SONUÇLAR

4.1. PERFORMANS METRİKLERİ VE DEĞERLENDİRME KRİTERLERİ

Performans metriklerini değerlendirmek üzere önerilen sistem tasarımında Grafana aracı kullanılmaktadır. Grafana açık kaynak yazılımı, herhangi bir lokasyonda saklanan metrikleri, sistem kayıtlarını, sorgu izlerini ve görselleştirmeleri sağlar. Grafana, verileri zaman serisi veritabanı (TSDB) ile anlaşılır görselleştirmelere dönüştürmek için araçlar sağlamaktadır [18].

Grafana, katkıda bulunan kişilerin ve resmi olarak yayımlanan tanımında olduğu gibi sistem performansını izlemek, analiz etmek için birçok metrik ve görselleştirmeler sunmaktadır. Önerilen sistem tasarımının farklı broker'ların sistemde ne kadar bant genişliği, bellek kullanımı, işlemci kullanımı, disk kullanımı gibi metrikleri elde etmemizi ve önerdiğimiz

sistemin ve araştırmacının amacını ortaya koyar. Karşılaştırma metriklerinin de elde edilmesini sağlamaktadır.

Değerlendirme kriterleri olarak, Grafana üzerinde bant genişliği kullanımı, bellek kullanımı ve CPU kullanımı gibi değerler ele alınır. IoT cihazından gelen verileriyle kayıt süreleri ele alınıp, işleme süresi kriterleri çerçevesinde de önerilen sistemin hangi broker üzerinde daha performanslı çalıştığını belirleme hedefimize yardımcı olacak Görsel 2'de görülen değerler çıkarılmıştır.

4.2. PERFORMANS KARŞILAŞTIRMALARI

Bu bölümde, farklı Message Broker tasarımlarının performanslarını karşılaştırmak için kullanılan metrikler ve elde edilen sonuçlar sunulmaktadır. Elde ettiğimiz sonuçlara göre en çarpıcı sonuç, broker sistemlerinin paralel çalışma sayısı, gelen verideki hesaplama işlemleriyle birlikte veritabanına erişim noktasında işleme süresinin (TimeToFindNearPoint) verilen konfigürasyonlara göre darboğaz yaratacak şekilde etki ettiği tespit edilmiştir.

4.2.1. KARŞILAŞTIRMADA KULLANILAN METRİKLER

Tablodaki her bir sütunun açıklamasını birer cümle ile ifade etmek gerekirse:

Id: Brokerin işlemesi üzere rasgele oluşturulan lokasyon bilgisinin benzersiz numarası.

City Name: Verilerin hangi şehre ait olduğunu belirten bir bilgi.

Neighborhood Name: Şehir içindeki belirli bir mahalle veya semtin adını gösteren bir bilgi.

Latitude: Belirli bir konumun kuzey-güney eksenindeki koordinatını temsil eden bir sayısal değer.

Longitude: Belirli bir konumun doğu-batı eksenindeki koordinatını temsil eden bir sayısal değer.

Created Date: Verinin oluşturulma tarih ve saat bilgisini gösteren zaman damgası.

Broker Type: Verinin, hangi broker teknolojisi kullanılarak alındığını veya yayımlandığını belirten bir bilgi.

Near Point Id: Rastgele oluşturulan konum bilgisinin yakınlık gösterdiği diğer Lokasyon Id bilgisi.

Time to Find Near Point Id: Rastgele oluşturulan bir konum bilgisinin diğer konuma ne kadar uzaklıkta olduğunu ve Yakınlık Id değerinin ne kadar zamanda işlenerek bulunduğunu gösteren milisaniye cinsindeki zaman bilgisi.

Test Group Id: Karşılaştırma esnasında toplu veri gönderimindeki senaryoların ayrıştırılması için kullanılan test gruplarının benzersiz alfa numerik değeri.

4.2.2. KARŞILAŞTIRMA ORTAMI VE TEST SENARYOLARI TANIMLAMA

Karşılaştırma işlemlerinde; veri işleme süresi, hafıza kullanımı, CPU kullanımı gibi metriklere kullanılan sistemin özellikleri de etkili olacağından 32 GB Hafıza, 12th Gen Intel(R) Core(TM) i7-12700H 4.7GHz, 5000mb's okuma/yazma hızı olan SSD sistemi üzerinde önerilen broker sistemi entegre edilmiştir. Karşılaştırma işlemlerinde broker sistemlerinin işleme süresi ve diğer kullanım metriklerinin diğer broker sistemleriyle yalıtılması için çekirdek uygulamamızda Docker Container sisteminin Compose dosyası ile "BROKERTYPE" parametresi ile test ve broker sistemi yalıtım sağlanarak, elde edilecek metriklerin bağımsız ve güvenilir olması amaçlanmıştır. Her bir broker tipindeki test süreçlerinin güvenilir ve doğru sonuç vermesi amacıyla broker sistemlerince okunan her bir lokasyon verisinin diğer lokasyon verilerine olan uzaklığını hesaplayacak olan enlem ve boylam verilerine en yakın olan lokasyonun bulunma algoritmasının işlem süresi test sürecindeki "Time to Find Near Point Id" kolonu olarak belirtilmiştir. Bu algoritmanın amacı, her bir mesajın broker sistemlerindeki işlenme süresine ne kadar etki ettiğini bağımsız bir parametreyle ölçmeyi amaçlamaktadır. Ayrıca her bir broker sisteminin aynı veritabanı üzerinde farklı zaman dilimlerinde işlem göreceğinden elde edilecek metriklerin broker sistemlerinin veri işleme ve maksimum transfer edebileceği mesaj sayısının etkilenmemesi üzere yapılan 1. Tip ve İşleme süresi karşılaştırması sırasında Cassandra veritabanı 2 kopya olarak çalıştırılmıştır. Ayrıca IoT sistemlerini taklit edebilmek amacıyla Apache JMeter test aracı kullanılarak Thread Group sistemiyle aynı anda çekirdek uygulamaya erişen IoT cihazı sayısını belirterek aynı IoT cihazından aynı anda gelecek olan veri sayısını 10 adet olmak üzere, saniyede bir 10 veri göndermesi için "Number Of Threads" sayısını 10, "Ramp-Up Periods" değerini 1 olarak sabit tutulmuştur.

Farklı broker sistemlerinin karşılaştırılması amacıyla 4 farklı test senaryosu tanımlanmıştır;

1- Broker tipi ve her bir IoT verisinin işlenme süresi
2- Broker kopya sayısı ve maksimum mesaj transfer sayısı
3- En performanslı 1.test sonucunu veren broker sisteminin Cassandra kopya sayısının arttırılarak, veritabanı kopya sayısının broker sistemi üzerine etkisi
4- Broker çekirdek kodunun Thread yordamıyla paralel çalıştırılmasıyla mesaj işlenme süresine ve maksimum transfer sayısına etkisi

Ayrıca yukarda tanımlanan test senaryolarının, broker tasarımlarının konternizasyon etrafında her bir test numarasının Grafana ile bant genişliği, hafıza kullanımı, CPU kullanımı bazında kullanım metriklerini sunarak en performanslı ve kullanılabilir broker sistemine ulaşmak istenmektedir.

4.3. PERFORMANS KARŞILAŞTIRMA SONUÇLARI

Çizelge 1. Broker tipi ve her bir IoT verisinin işlenme süresi

Broker Type	***Near Point Find Avarage / Yakınlık Derecesi Bulma Hız Ortalaması***	***Broker Core Application Replication / Ana broker uygulamasının kopya sayısı***	***Broker Application Replication / Broker uygulamasının kopya sayısı***
ACTIVEMQ	171 Milisaniye	1	1
RABBITMQ	486 Milisaniye	1	1
KAFKA	260 Milisaniye	1	1

Her bir broker tipine gerçekleştirilen 10.000 adet sahte olarak üretilen lokasyon bilgisinin üretilme ve broker'a aktarılması üzere çekirdek uygulamada tanımlanan "BROKER_TYPE" ortam değişkeni ile her bir broker testi için olmak üzere tanımlamalar gerçekleştirilerek Çizelge 1'deki sonuçlar elde edilmektedir. Ayrıca çekirdek uygulama üzerinde tanımlanan her bir ortam değişkeninin açıklaması açık kaynak olarak ReadMe marked file olarak proje üzerinde paylaşılmıştır [18].

IoT vb. cihazlardan gelecek lokasyon verilerinin yapay zekâ modüllerinde veya LLM modüllerinde işletilmek gibi işlemler bulunmaktadır. Önerilen bu sistem de bu işlemleri taklit etmek üzere, lokasyon yakınlık bulma süre ortalaması (Near Point Find Avarage) değeri oluşturulmuştur. Bu değer ile her bir IoT verisinin işlenme süresi ile önerilen sistemin performansı sunulmaktadır. RabbitMQ broker modelinde verinin kuyruktan alınma ve işlenme süresinin diğer broker sistemlerine göre oldukça uzun işlem süresi verdiği gözlemlenmektedir. Maksimum performansta mesaj transfer sayısına nazaran ActiveMQ broker sisteminin, RabbitMQ broker sisteminden verilere daha hızlı erişim sunduğu belirlenmiştir.

Çizelge 2. Broker kopya sayısı ve maksimum mesaj transfer sayısı

Broker Type	***Broker Kopya Sayısı***	***Maksimum Performansta Mesaj Transfer Sayısı***
ACTIVEMQ	1	82.000
RABBITMQ	1	100.000
KAFKA	3	310.000

Çizelge 3. Broker kopya sayısı ve maksimum mesaj transfer sayısı

Broker Type	***Broker Kopya Sayısı***	***Maksimum Performansta Mesaj Transfer Sayısı***
ACTIVEMQ	2	95.000
RABBITMQ	2	120.000
KAFKA	4	360.000

Her bir broker'da maksimum mesaj transfer sayısının sınırı olmamakla birlikte, broker sistemlerindeki yüke bağlı olarak mesajların transfer edilebilme süreleri uzayabilmektedir. Bu süre broker tasarımlarının protokolleri, tasarım kalıplarına, kuyruk modellerine, kuyruğun öncelik türlerine vb. göre değişebilmektedir. Bu karşılaştırma ile önerilen broker sistemlerinin ilerisine gitmeyerek artan mesaj hacmine göre brokerların mesajları okuma sürelerine göre maksimum performansta alabileceği mesaj transfer sayısı belirlenmiştir, bu belirlemede brokerların sunduğu yönetim panellerinden yararlanılmıştır.

Çizelge 4. En performanslı 1.test sonucunu veren broker sisteminin cassandra kopya sayısının arttırılarak, veritabanı kopya sayısının broker sistemi üzerine etkisi

Best Effort Broker Type	***Cassandra Replication***	***Near Point Find Avarage***
ACTIVEMQ	3	98.57 Milisaniye

Cassandra'nın veritabanı uygulamasının kopya sayısı arttıkça işlenme süresinin düşmesi, yükün daha iyi dağıtılarak işlem yükünün azaltılmasına olanak tanır. Bu, Cassandra'nın veriyi Partition şeklinde tutmasından kaynaklanır. Her bir kopya, farklı kolonlardaki veriyi gönderdiğinden dolayı, daha hızlı işlem yapabilmesine olanak sağlar. Ayrıca Cassandra'nın diğer NoSQL veritabanlarından farklı olarak tek bir "Writer Node" olmaması ve her kopyanın hem yazma hem de okuma görevi alabilmesi, yazma işlemlerinde gecikme yaşanmadan her kopyaya yazma işlemi yapılabilmesini sağlar.

Bu özellikler, işlem yükünün daha etkin bir şekilde dağıtılmasını ve her bir kopyanın veri işleme sürelerinin azalmasını sağlar. Sonuç olarak, Cassandra'nın veritabanı kopya sayısının artması, genel işlem sürelerinin düşmesine ve daha yüksek bir performans elde edilmesine katkı sağlar. Bu özellikler, dağıtık veritabanı sistemlerinde veri erişiminde ve işlemlerde daha hızlı ve etkili bir performansın sağlanmasına yönelik önemli avantajlar sunar.

Çizelge 5. Broker çekirdek kodunun Thread yordamıyla paralel çalıştırılmasıyla mesaj işlenme süresine ve maksimum transfer sayısına etkisi

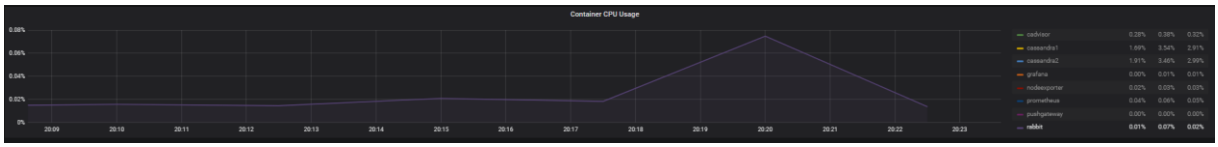
Broker Type	***Near Point Find Avarage / Yakınlık Derecesi Bulma Hız Ortalaması***	***Broker Core Application Replication / Ana broker uygulamasının kopya sayısı***	***Broker Application Replication / Broker uygulamasının kopya sayısı***	***Message Count / Mesaj Transfer Sayısı***
ACTIVEMQ	140 Milisaniye	2	1	10.000
RABBITMQ	376 Milisaniye	2	1	10.000
KAFKA	214 Milisaniye	2	1	10.000

Broker çekirdek kodunun Thread aracılığıyla paralel çalıştırılmasının mesaj işlenme süresine ve maksimum transfer sayısına etkisi incelendiğinde, elde edilen metriklerde belirgin bir iyileşme görülmektedir. Ana broker uygulamasının, yani kuyrukların Consume edildiği uygulamanın çokluğuna bağlı olarak paralel işlem yapılması, işlenme süresinin azalmasını sağlamaktadır.

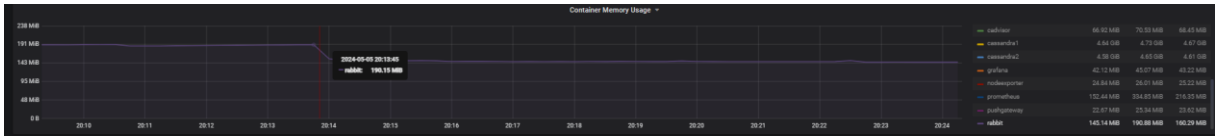
Bu durum, çekirdek kodunun aynı anda birden fazla iş parçasını işleyebilmesinden kaynaklanmaktadır. Her bir mesajın ayrı bir Thread içinde işlenmesi, işlem sürelerinin paralel olarak yürütülmesine imkân tanır. Bu da toplam işlenme süresinin azalmasını sağlar.

Ayrıca maksimum transfer sayısında da belirgin bir artış gözlemlenmesi beklenir. Ancak hız ortalamasının değişimine odaklanıldığından bu parametre sabit tutulmuştur. Artış gözlemlenmesinin sebebi ise paralel işlem yapıldığında, daha fazla mesaj aynı anda işlenebilir ve transfer edilebilir hale gelir. Bu da sistemdeki maksimum transfer kapasitesinin artmasını sağlar.

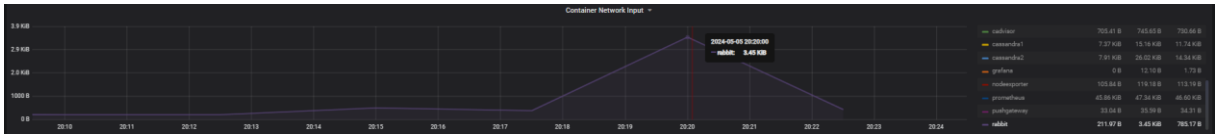
Test senaryolarının sistem kullanım metrikleri Grafana aracılığıyla görsel olarak elde edildikten sonra verilerin sayısallaştırılarak karşılaştırmaya uygun hale Çizelge 6'daki gibi getirilerek günümüz sistemlerindeki IoT ortamlarına en uygun broker sistemine ulaşılmasında kullanılan donanım metriklerinin elde edilerek önerilen sistemin en düşük kullanımda en yüksek kapasiteyi sunduğunu göstermek amaçlanmıştır.



Görsel 3. Önerilen Sistemlerin Memory - CPU - Bant Genişliği Kullanımı, Grafana ile



Görsel 4. Önerilen Sistemlerin Memory - CPU - Bant Genişliği Kullanımı, Grafana ile



Görsel 5. Önerilen Sistemlerin Memory - CPU - Bant Genişliği Kullanımı, Grafana ile

Çizelge 6. Sistem Kullanım Değerleri

Tablo Test Numarası	***Broker Tipi***	***Bant Genişliği Kullanımı (Avg.)***	***Hafıza Kullanımı(Avg.)***	***CPU kullanımı(Avg.)***
1	ACTIVEMQ	14.62 Kib	554.60 Mib	%1.05
1	RABBITMQ	556.53 Kib	179.30 Mib	%0.02
1	KAFKA	575.39 Kib	1.712 Mib	%0.28
3	ACTIVEMQ	28.33 Kib	680.38Mib	%1.96
3	RABBITMQ	953.68 Kib	245.34 Mib	%0.05
3	KAFKA	895.78 Kib	2.485 Mib	%0.36
4	ACTIVEMQ	15.42 Kib	542.71Mib	%1.04

Yapılan karşılaştırmalı testler sonucunda, anlık veri işleme gerekliliği bulunan IoT sistemlerine, yoğun veri akışı gerçekleştiren sonradan işlemeli IoT sistemlerine ve potansiyel asenkron mesajlaşma süreçlerinde kullanılabilecek broker sistem tasarımının yatayda ölçeklenebilirlik sağlayan Docker ve yönetme sağlayan K8s aracılığıyla artan hacimli verilerin hem işlenme süresi çerçevesinde hem de broker sistemlerinin transfer edebileceği maksimum veri sayıları üzerinde anlamlı sonuçlar elde edilmiştir. Tablolara referans edilen metrik değerleri public şekilde üzerinde paylaşılmıştır [19].

Sonuçlara göre, her test senaryosunun kopya sayısı arttıkça maksimum transfer edebileceği veri sayısını arttırabileceği, işlenme sürelerinin ortalamalarının düştüğü gözlemlenmiştir. Önerilen sistem çerçevesinde çekirdek uygulamanın da hem Spring Framework destekli hem de uygulamayı geliştirici tarafından geliştirilen Reactive Programming mekanizmalarıyla paralel çalıştırılması sayesinde önerilen sistemin önemli ölçüde performansına olumlu etki ettiği gözlemlenmiştir. Amaçlanan yüksek performanslı ve genişletilebilir broker sistem önermesinin yarattığı sistem kullanım metrikleri de değerlendirilerek önerilen sistemlerin hangi donanım metriklerine etki ettiğine dair elde edilen Çizelge 6'daki değerler ile önerilen sistemlerin çalıştırılabilecek ortamların gereksinimlerini ortaya çıkartmıştır.

Önerdiğimiz sistem tasarımında, RABBITMQ broker sisteminin diğer broker sistemlere göre daha hızlı ve daha çok sayıda veri işleyebildiği verilen sonuçlara göre elde edilip, sistem gereksinimlerini daha az kullandığı ve diğer sistemlerden işleme performansı/gereksinim değerleri bakımından öne geçtiği tespit edilmiştir.

Bu çalışmanın bulguları neticesinde, daha önce literatürde Yasin Görmez vd. tarafından ele alınan "Efficient and Scalable Broker Design for the Internet of Things Environments" makalesinde [20] önerilen broker sistemini, iyileştirmekte ve daha performanslı ve veri kaybına dayanıklı, veriye erişimde daha hızlı sistem tasarımlarının önerilebileceği kanıtlanmıştır.

5. SONUÇ VE GELECEKTEKİ ÇALIŞMALAR

Bu çalışma, Nesnelerin İnterneti (IoT) alanında önemli bir sorunu ele almış ve farklı broker tasarımlarının performansını incelemiştir. Elde edilen sonuçlar, birçok bilimsel araştırmaya katkı sağlayacak niteliktedir. Çeşitli broker tasarımlarının işlem süreleri üzerindeki etkisini net bir şekilde göstermiştir. Bu tasarımların performansını karşılaştırmak için analitik ve etkileşimli bir yöntem kullanılmıştır. Grafana aracılığıyla sunulan görselleştirmeler yapılmıştır. Bu sayede de farklı broker tasarımlarının anlamlı bir şekilde değerlendirilmesi sağlanmıştır.

Gelecek çalışmalar açısından, bu araştırma alanındaki eksiklikler ve potansiyel geliştirmeler üzerinde odaklanılabilir. Örneğin, daha fazla broker tasarımı incelenebilir ve farklı senaryolarda performansları değerlendirilebilir. Daha fazla veri alınabilir ve aynı anda karşılaştırmalar gerçekleşebilir. Ayrıca, gerçek dünya uygulamalarında bu broker tasarımlarının nasıl performans gösterdiğine dair daha fazla veri toplanabilir. Bununla birlikte, bu çalışma NoSQL veritabanı üzerinde yapılan analizlerle sınırlı kalmıştır, bu nedenle ilişkisel veritabanları ve diğer veri depolama yöntemlerinin incelenmesi de gelecek araştırmalarda değerlendirilebilir.

Sonuç olarak, bu çalışma IoT alanında önemli bir probleme odaklanmış ve farklı broker tasarımlarının performansını karşılaştırarak bu alandaki araştırmalara katkıda bulunmuştur. Gelecek çalışmalar, bu alandaki bilgiyi genişleterek ve yeni, güçlü teknolojilere odaklanarak bu araştırmanın etkisini artırabilir.

KAYNAKÇA

[1] https://aws.amazon.com/tr/what-is/iot/ - (Erişim Tarihi: 01.05.2024)

[2] https://acikerisim.sakarya.edu.tr/bitstream/20.500.12619/79673/1/T08082.pdf - (Erişim Tarihi: 05.05.2024)

[3] https://gsl.com.tr/mqtt-endustriyel-nesnelerin-interneti-uygulamalarinda-verimli-ve-esnek-bir-haberlesme-yapisi.html - (Erişim Tarihi: 01.05.2024)

[4] https://aws.amazon.com/tr/what-is/mqtt/ - (Erişim Tarihi: 01.05.2024)

[5] https://www.tercihyazilim.com/blog/http-nedir - Erişim Tarihi: 25.04.2024.

[6] https://appmaster.io/tr/blog/websockets-nedir-ve-nasil-olusturulur - Erişim Tarihi: 25.04.2024.

[7] https://www.hosting.com.tr/bilgi-bankasi/http-2-nedir-ne-ise-yarar/ - Erişim Tarihi: 25.04.2024.

[8] https://prodiot.com/tr/coap-protokol-ozellikleri-nedir-/article/6 - Erişim Tarihi: 26.04.2024.

[9] https://www.ozztech.net/genel/message-broker-nedir-2/ - Erişim Tarihi: 05.05.2024

[10] https://www.ariteknokent.com.tr/tr/haberler/milsoft-seminerine-davetlisiniz - Erişim Tarihi: 26.04.2024.

[11] https://tr.wikipedia.org/wiki/XMPP - Erişim Tarihi: 26.04.2024.

[12] https://www.rabbitmq.com/ - Erişim Tarihi: 26.04.2024.

[13] https://kubernetes.io/ - Erişim Tarihi: 05.05.2024

[14] https://activemq.apache.org/ - Erişim Tarihi: 26.04.2024.

[15] https://kafka.apache.org/ - Erişim Tarihi: 26.04.2024.

[16] https://cassandra.apache.org/_/index.html - Erişim Tarihi: 26.04.2024.

[17] https://docs.docker.com/build/cloud/ - Erişim Tarihi: 05.05.2024

[18] https://grafana.com/docs/grafana/latest/introduction/ - Erişim Tarihi: 05.05.2024

[19] https://github.com/sogz/bigdatabrokersystem/ - Erişim Tarihi: 07.05.2024

[20] Y. Görmez, H. Arslan and Ö. F. Kelek, "Efficient and Scalable Broker Design for the Internet of Things Environments," 2020 28th Signal Processing and Communications Applications Conference (SIU), Gaziantep, Turkey, 2020 Erişim Tarihi: 11.05.2024