IoT Cihaz Verileri İçin Gerçek Zamanlı ve Ölçeklenebilir Büyük Veri Mimarisi

Arif Kamil YILMAZ ve Devrim Barış ACAR

STM, Ankara, Türkiye
Akyilmaz@gmail.com, devrimbaris@gmail.com

Özet. Günümüzde IoT (Internet of Things - IoT) (nesnelerin interneti) cihazlarının giderek daha yaygın olarak kullanılmaya başlanması, bu cihazlardan gelen verinin işlenmesi ve saklanması için farklı yaklaşımların kullanılması ihtiyacını doğurmuştur. Bunun sebebi bu cihazlardan gelen verinin çok büyük miktarlarda ve anlık olarak çok büyük hızlarda gelmesidir. Ek olarak bu uygulamaların çoğunda akan veri üzerinde karmaşık analizlerin gerçeğe yakın zamanlı olarak yapılmasına (karmaşık olay işleme - complex event processing) ihtiyaç duyulmaktadır. Bu konudaki diğer ihtiyaçlar; alınan verilerin uzun süreler boyunca saklanması, bu saklanan veriler üzerinde ihtiyaç duyulması halinde belirli performans kıstaslarına göre hızlı sorgular(ad hoc) yapılabilmesi, karmaşık analizler koşulması ve makine öğrenme ve yapay zekâ modelleri için temel veri seti olarak kullanılması olarak listelenebilir. Diğer bir bağlam olarak, günümüzde veri akış hızları ve yoğunluğu değişiklikler gösterebilmektedir. Sistemler ilk kurulduğunda düşük olan veri akışları, ek veri kaynaklarının devreye alınmasıyla büyüyebilmektedir. Tasarlanan sistemlerin bu yeni kaynakları da işleyebilmesi ve depolayabilmesi için yazılım sistemlerinin ölçeklenebilir bir mimari düşünülerek tasarlanması gerekmektedir. Bu çalışmada, her biri saniyede iki kez, 48 farklı ölçüm verisi yollayan 1000 IoT cihazından gelen verilerin, yukarda listelenen ihtiyaçlar doğrultusunda alınması, karmaşık olay işleme (CEP) teknikleri kullanılarak işlenmesi, saklanması ve sorgulanması için geliştirilen büyük veri sistem mimarisi ele alınmıştır.

Anahtar Kelimeler: Nesnelerin İnterneti, IoT, Karmaşık Olay İşleme, Büyük Veri, Mesaj Kuyruklama, NoSQL, Veri Tabanı, Internet Of Things, CEP, Complex Event Processing.

Real Time and Scalable Big Data Architecture for IoT Data

Arif Kamil YILMAZ and Devrim Barış ACAR

STM, Ankara, Turkey
Akyilmaz@gmail.com, devrimbaris@gmail.com

Abstract. The growing pervasive use of IoT (Internet of Things - IoT) devices has introduced the need of different approaches for processing and storing data from these devices. This was partially caused by the massive data from these devices and partially the speed of this data. In addition, there is a need for complex analysis on complex data streams (complex event processing) on the data flowing in the majority of these applications. Other needs in this regard are; the storage need of data for long periods of time, ad hoc query and complex analysis needs on the stored data, the remodeling of data for complex learning and artificial intelligence models. From another perspective, data transfer rates and density may vary over time. When the systems are first installed, data rates may be low but by the introduction of additional data sources data rates can grow exponentially. The system architecture should be designed in a scalable way, so that it can process and store data effectively. In this study, a large data system architecture was developed for handling, storing and querying data from 1000 IOT devices, each of which sends 48 different measurement data twice a second, according to the needs listed above, using complex event processing (CEP) techniques.

Keywords: Internet of Things, IoT, Complex Event Processing, Big Data, Message Queuing, NoSQL, Database.

1 Giriş

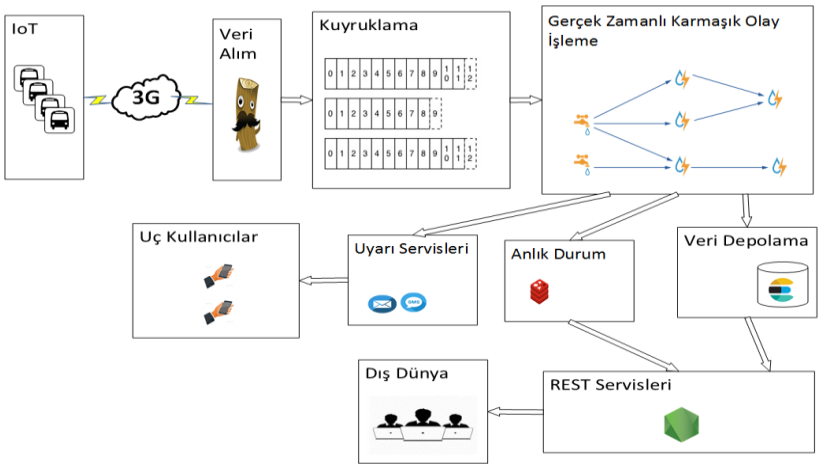
Nesnelerin interneti (IoT), gerçek dünyadaki nesnelerin internet ağı üzerinden diğer nesne ya da sistemlerle iletişim içinde olduğu dinamik evrensel bir network yapısı olarak tanımlanabilir. IoT cihazlar tarafından uç noktalarda üretilen olaylar (event) genellikle semantik olarak ilkel, tek başına çok fazla anlam içermeyen mesajlardır. Gerçek zamanlı olay işleme uygulamalarında, temel olarak ulaşılmak istenen bilgi, akan veri üzerinde iş mantığı ve kurallarının işletilmesi sonucundan elde edilen bilgidir [1]. Örnek olarak otobüs üzerindeki sensörlerden alınan anlık motor devri bilgisinin tek başına bir anlamı olmazken, 10 saniye boyunca bu bilginin yorumlanması ile şoförün aracı ekonomik kullanmadığı sonucuna ulaşılıp anlık olarak uyarı ve önlem mekanizmaları işletilebilir. Bu ve buna benzer birinci seviye sensör verilerinin işlenerek daha üst seviye değeri olan sonuçlar çıkarma işlemine karmaşık olay işleme (CEP) ismi verilmektedir.

Bu çalışmamızda gerçek zamanlı akan veri üzerinde 20 farklı olay türü için, karmaşık olay işleme teknikleri kullanılarak, aykırı durumlar tespit edilip uyarı mekanizmaları işletilmiştir. Akan veri sonsuz uzunluklu ve yüksek frekanslı olduğu düşünüldüğünde, bu veri kümesi üzerindeki anlık CEP işlemleri dağıtık ve genişleyebilir bir mimaride, paralel olarak ele alınmıştır. Veri kaybı yaşamamak ve 1000 farklı kaynaktan akan verinin sıralı ve paralel olarak CEP işlemle altyapısı tarafından tüketilebilmesi için dağıtık kuyruklama yapısı kullanılmıştır. Verinin IoT cihazlardan veri işleme altyapısına güvenli şekilde aktarılması için güvenli veri alım (secure data ingestion) katmanı tasarlanmıştır. Sensör verilerinin ve üretilen aykırı durumların performans kriterleri altında sorgulanabilmesi için dağıtık veri depolama

altyapısı kullanılmıştır. Son olarak depolama altyapısında tutulan veriler güvenli REST servisler üzerinden tüketicilere(consumer) açılmıştır.

2 Sistem Mimarisi

Şekil 1'de çalışmamızda kullanılan bileşenleri genel sistem mimarisi görünmektedir. Otobüsler üzerine takılan IoT cihazlarından 3G aracılığı ile aktarılan sensör verileri "Veri Alım" bileşeni tarafından alındıktan sonra konu (topic) bazlı çalışan "Kuyruklama" bileşenine aktarılır. Sistem otobüsler üzerinde halihazırda tanımlanmış otobüs kimlik numarasına (Vehicle İdentification Number) göre otobüsleri ayırt etmektedir. "Gerçek Zamanlı Karmaşık Olay İşleme" bileşeni, "Kuyruklama" bileşeninde kaynak türüne (otobüs kimlik numarası) göre bölümlendirilmiş veriyi paralel olarak tüketir. Sensör verisi üzerinde karmaşık olay işleme yapılırken aynı zamanda veriler zenginleştirilerek "Veri Depolama" ve "Anlık Durum" bileşenlerine aktarılır. Tespit edilen aykırı durumlar "Uyarı Servisleri" bileşeni tarafından email ve sms olarak uç kullanıcılara bildirilir ve "Rest Servisleri" aracılığı ile depolanan veriler diğer uç birimler tarafından tüketilmek üzere dış dünyaya açılır.



Şekil 1. Genel Sistem Mimarisi

3 Bileşenler

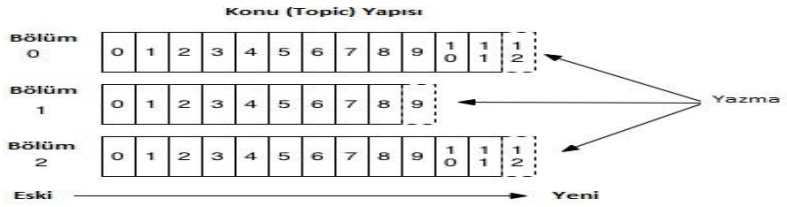
3.1 Veri Alım

Veri alım bileşeni [2] aracılığı ile 1000 farklı otobüsten eş zamanlı olarak gelen sensör verileri TCP protokolü kullanılarak alınmıştır. Sisteme akan verinin güvenilir kaynaklardan ve şifreli şekilde gelmesi için IoT cihazlar ile veri alım bileşeni arasında SSL/TSL protokolü kullanılmıştır. Merkezde üretilen güvenlik sertifikaları uç birimlere dağıtılmış böylece uç birimlerdeki cihazların kimlik doğrulamaları ve verinin şifreli

olarak iletilmesi sağlanmıştır. Alınan sensör verisi, içerisindeki kaynak kimlik bilgisine (otobüs kimlik numarası) göre kuyruklama yapısındaki ilgili bölüm (partition) ve ilgili konu(topic) kuyruğuna yönlendirilmiştir.

3.2 Kuyruklama

Kuyruklama bileşeni, dağıtık mimaride çalışan, eş zamanlı olarak veri alım bileşeninden gelen sensör verilerini konu bazlı olarak, birden fazla bölümde(partition) tutan bir yapıya sahiptir. Kuyruklama bileşeni olarak açık kaynaklı Apache Kafka [3] platformu kullanılmıştır. Tutulan her kayıt anahtar, değer ve zaman damgalarını içerir. Şekil 2' deki kuyruklama yapısında, her konuya ait veriler konfigure edilebilen sayıda bölüm üzerinde tutulur. Veriler bölümlerde sıralı olarak bulunur. Bölümlerdeki her kayda sıra numarası atanır ve bu sıra numarası offset olarak isimlendirilir. Bu offset numarası bölümdeki her kaydın tekil/eşsiz olarak adreslenmesini sağlar. Veri alım bileşeni tarafından gelen veriler, Şekil 2'de "Yazma" olarak gösterilen yönde, veri içerisindeki otobüs kimlik numarasına göre ilgili bölüme aktarılmıştır. Kuyruklama yapısındaki her kayıt, belli bir süre tüketiciler(consumer) tarafından kullanılmak üzere saklanır. Bu süre dolduktan sonra kayıtlar kuyruklama yapısının kullanım alanını artırmak için dışarı atılır. Kuyruklama yapısında kullanılan bölüm yapısının 2 önemli avantajı vardır. Birincisi, gelen verilerinin tek bir sunucuya sığacak boyutun ötesinde ölçeklendirmeye izin vermesi, ikincisi ise bölümlerin ondan veri alan tüketiciler için paralellik birimi olarak çalışıyor olmasıdır. Çalışmamızda otobüslerden alınan sensör verileri, dağıtık olarak 3 adet sunucu üzerinde çalışan kuyruklama yapısı üzerinde, tek bir konu ile ilişkilendirilerek tutulmuştur. Bu konuya ait veriler için 8 adet bölüm(partition) oluşturulmuştur. Sonraki bileşen olan "Gerçek Zamanlı Karmaşık Olay İşleme" bileşeni tarafından 8 tüketici kanalı üzerinden kayıtların tüketilmesi sağlanmıştır.

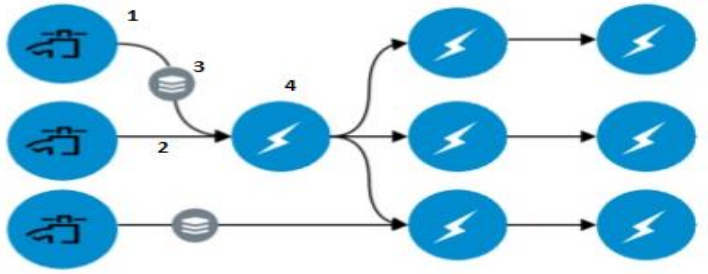


Şekil 2. Kuyruklama Bileşeni Yapısı

3.3 Gerçek Zamanlı Karmaşık Olay İşleme

Kuyruklama bileşeninden otobüs kimlik numarasına göre bölümlendirilmiş verileri alarak dağıtık mimaride işleyen bileşendir. Bu bileşenin gerçekleştirimi için açık kaynaklı dağıtık veri işleme platformu olan Apache Storm [4] kullanılmıştır. Şekil 3' de "Gerçek Zamanlı Karmaşık Olay İşleme" bileşenin genel yapısı gösterilmiştir. Bu yapıdaki tüm bileşenlerin ortaya çıkardığı yapıya gerçek zamanlı veri işleme topolojisi

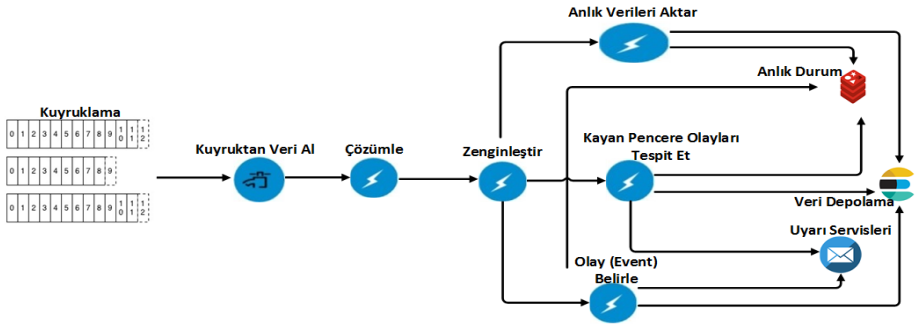
denir. Şekil 3'de 1 ile belirtilen alt bileşen veri kaynaklarından verinin alınmasından sorumludur ve Spout olarak isimlendirilir. 2 ile belirtilen alt bileşen sistem içerisinde akan veriyi(stream) temsil eder. 3 ile gösterilen yapı ise bileşenler arasından akan birim veri yani Tuple olarak tanımlanır. Akan veri üzerinde topoloji boyunca verilerin işlenmesi ise 4 numaralı alt bileşen ise Bolt tarafından yapılır. Topoloji boyunca yapılan gruplama, birleştirme, filtreleme, veri tabanlarıyla iletişim vb. tüm işlemler Boltlar tarafından gerçekleştirilir.



Şekil 3. Gerçek Zamanlı Karmaşık Olay İşleme Bileşenleri

Şekil 4'de çalışmamızda gerçekleştirdiğimiz topolojinin yapısı görünmektedir. 3 sunucuda 8 işlem (thread) olarak çalışan "Kuyruktan Veri Al" bileşeni tarafından verilerin topolojiye alınması sağlanır. "Kuyruktan Veri Al" bileşeni her 5 saniyede bir bulunduğu offset bilgisini günceller. Böylece topolojinin yeninden yüklenmesi, bağımlı olan diğer bileşenlerde hata durumları oluşması vb. durumlarda veri kaybı yaşanmadan tüm mesajların işlenmesi sağlanır. Mesajların en az bir kez işlenmesinin garanti edilmesi için topolojideki uç bileşenlerden ilgili mesajla ilgili başarılı olarak işlendiğine dair geri bildirim (ack) mesajının gelmesi gerekmektedir. Aykırı durumlar oluştuğunda "Kuyruktan Veri Al" bileşeni Kuyruklama bileşenindeki son kaldığı offset bilgisini kullanarak mesajları yeniden tüketmeye devam eder. Topoloji yeniden başladığında herhangi bir offset bilgisi yok ise gelen mesajlar kuyruğa son gelenden itibaren işlenmeye devam edilir. Kuyruktan alınan CSV (comma separated value) formatındaki sensör verileri parçalanarak mesaj alınları çözümlenir. Mesaj içindeki otobüs kimlik bilgisi anahtar alan olarak kullanılarak aynı otobüse ait mesajların topolojide aynı boltlar tarafından işlenmesi sağlanır. Sensör verilerindeki aykırı durumlar tespit edilirken 2 farklı tür bolt kullanılmıştır. Biri kayan pencere mantığına göre belirli bir süre mesajları gruplayıp işleyen, diğeri ise her gelen mesaj alanındaki aykırı durumları belirli kurallar çerçevesinde tespit eden boltdur. Bu iki tür bolt içinde sistem genelinde daha önce tanımlanmış kurallar işletilerek aykırı durumlar tespit edilmiştir. Her aykırı durum tespitinde kullanılan bolt bileşeniyle eşleşmiş bir thread bellek odaklı (in-memory) çalışan bir veritabanındaki belirli bir konuyla ilgili değişime abone(subscribe) olmuştur. Bu konuya herhangi bir değişim mesajı (publish) gönderildiğinde bu konuyu dinleyen thread uyarılıp, yeni yayımlanan kural bilgisi ilgili boltlarda çalışma zamanında güncellenmiştir. Böylece aykırı durumların tespit edilmesinde kullanılan kurallar çalışma zamanında dinamik olarak değiştirilmiş ve topolojinin yeniden başlatılmasına ihtiyaç duyulmamıştır. Benzer işlevleri yerine getiren altyapılar

incelendiğinde çalışma zamanında kural bilgilerinin dinamik olarak değiştirilebilmesi çalışmamızda bulunan farklı bir yaklaşım olarak göze çarpmaktadır. Aykırı durumların tespit edilmesinden sonra bu aykırı durumlar diğer bileşenler olan "Uyarı Servisleri" , "Anlık Durum" ve "Veri Depolama" bileşenlerine aktarılmıştır. Topolojide bulunan bileşenler farklı paralellik seviyelerinde 3 makinalı dağıtık bir yapıda çalıştırılmıştır.



Şekil 4. Gerçek Zamanlı Karmaşık Olay İşleme Topolojisi

3.4 Uyarı Servisleri

Anlık olarak tespit edilen aykırı durumlar publish/subscribe yapısında çalışan bellek tabanlı veritabanına aktarılmıştır. "Uyarı Servisleri", bellek tabanlı veri tabanı üzerindeki ilgili konuya abone olarak herhangi bir aykırı durum oluştuğunda bu aykırı durumu dağıtmakla sorumlu bileşendir. Sistemde ön tanımlı olarak her bir aykırı durum tipi için gönderim sıkılığı ve şekli tanımlanmıştır. Uyarı servisleri tarafından, anlık olarak oluşan aykırı durum tipine göre belirtilmiş sıklıkta, email veya sms olarak uyarılar uç kullanıcılara gönderilir.

3.5 Veri Depolama

Veri depolama altyapısının seçimi esnasında aşağıda listelenen kıstaslardan faydalanılarak karşılaştırma yapılmıştır.

- Analitik sorgulara hızlı cevap verebilmesi
- Artan kapasite ihtiyaçlarına göre kolayca ölçeklendirilebilmesi
- Hataya karşı dirençli olması (fault tolerant)
- Kolay yapılandırma ve yönetim

Bu kıstaslara göre Apache Lucene [5] tabanlı, dağıtık çalışan Elasticsearch depolama alt yapısı seçilmiştir. Bu altyapı; veriyi, bölümlendirilmiş (shard) ve her bölümü farklı kopyalar şeklinde (replication) diğer makinalarda paylaşarak saklamaktadır. Bu bölümlendirilmiş yapıyla beraber sorgu ve veri kayıt işlemleri dağıtık bir şekilde gerçekleştirilmektedir. Kopyalar sayesinde ise verilerin tutulduğu makinede bir arıza olması durumunda, kopyaların bulunduğu makine otomatik olarak devreye girerek sorgulara ve veri kayıt isteklerine cevap vermektedir.

Sistemi işletecek kurum tarafından verilerin silinmemesine yönelik istere istinaden, yıllara göre verinin ne kadar büyüyeceğine dair tahminleme yapılmıştır. Bu tahminlemede aşağıdaki parametreler kullanılarak hesaplama yöntemine gidilmiştir.

- İlk etapta devreye alınacak telemetri cihazı sayısı
- Takip eden aylarda aylık ne kadar cihaz ekleneceği
- Saniyede yollanan veri sayısı
- Cihazların günlük çalışma süreleri
- Verilerin depolama sisteminde kaç kopya olarak tutulacağı
- Sistemin önerilen maksimum doluluk oranı (%75 olarak sabitlenmiştir)

Sistemin 100 telemetri cihazı ile devreye alınması ve 3 yıl içinde 1000 cihaza çıkacağı öngörülmüştür. Her cihazın saniyede 2 veri yollayacağı, cihazların günün 18 saati boyunca çalışacağı ve her verinin yaklaşık 50 alan olacağı varsayılmıştır.

Sistemi işletecek kurumun internet üzerinden ücretli olarak bulut servisleri kullanma gereksinime göre, masrafların azaltılması amacıyla, makinaların sistem isterlerini sağlayacak minimum donanım seviyesinde yapılandırılması amaçlanmıştır. Bunun sonucunda ilk 5 ay ve 100 telemetri cihazı için depolama alt yapısı 2 makine yapılandırılarak sağlanmıştır.

Depolama sisteminin altyapısal olarak bazı master makinalara gereksinim duymaktadır. Bu makinalar sayesinde makinaların ağ sorunu vb. sorunlar nedeniyle sistemden erişiminin kesilmesi durumunda, yapının tek ve bütüncül olarak hizmet vermesi sağlanmaktadır. Örnek olarak, yukarda bahsedilen 2 depolama makinası ve ek olarak 1 makine master adayı makine yapılandırılmıştır. Sistem ayağa kaldırılması sırasında bütün depolama makinalarının en azından 2 master adayı makinayı görme zorunluluğu koyulmuştur. Buna göre depolama makinalarından birinin ağ bağlantısı kesilirse tek başına sadece kendini master adayı olarak görebileceği için otomatik olarak isteklere cevap vermeyecektir. Diğer makine ise hem master adayı makinayı hem de kendini göreceği için isteklere cevap verecektir. Sistemde makinaların görmesi gereken master makine sayısı şu formule göre hesaplanır:

$$(\text{master adayı makine sayısı} / 2 + 1)$$

Bahsedilen master makine gereksinimleri sebebiyle depolama kümesi en az 2 master adayı görmesi gerekecek şekilde yapılandırılmıştır. Toplam master adayı makine sayısı 3'dür.

3.6 Sistem Alt Yapısı ve Test Ortamı

Sistem; web sunucusu, web sunucu yük dengeleyicisi, mesaj toplayıcı, mesaj kuyruklama sistemi, mesaj işleme ,bunlara destek olacak yardımcı servis (zookeeper) ve depolama sistemi olmak üzere 7 servisten oluşmaktadır. Tüm sistem parçaları bir bulut sağlayıcı üzerinde konuşlandırılmıştır. İlk aylar için 9 makina bu servislerin koşması için ayarlanmıştır.

Tüm bu servislerin geliştirici tarafından yerel bilgisayarlarda da test amaçlı olarak koşturulması gerekmektedir. Bu amaçla Docker Container [6] altyapısı oluşturulmuştur. Bu altyapıyla tüm bu servisler ve versiyonları izole bir biçimde geliştiricinin ve test sorumlusunun bilgisayarında ayağa kaldırılmaktadır. Bu sayede yerel bilgisayarlarda ayar yapılmamakta, tüm altyapı ayarları ve bağımlılıklar tek dosya üzerinden tamamlanarak sistem çalıştırılmaktadır. Git versiyon kontrolünden son halini alan kullanıcı, sadece tek komut çalıştırarak güncel versiyonu ayağa kaldırmaktadır.

Son yıllarda bulut sistemleri üzerinde de benzeri altyapıları ayağa kaldırmak için konfigürasyon yönetim araçları giderek daha da artan şekilde kullanılmaya başlanmıştır. "Infrastructure as Code" kavramı kapsamında, altyapılar bilindik kod parçaları ile tanımlanmaktadır. Buna örnek olarak Vagrant sanallaştırma altyapısı verilebilir. Ruby kodu üzerinden makineler tanımlanarak altyapı tanımlanmakta ve çalıştırılmaktadır. Özetle, bu çalışma kapsamında aşağıdaki altyapılar sistem testleri için kullanılmaktadır.

1. Geliştirici bilgisayarlarında altyapının ayağa kaldırılması için Docker ve docker-compose
2. Test ortamının bulut sağlayıcı ortamında ayağa kaldırılması için Ansible
3. Test ortamının ayarlarının yerel bilgisayarlarda ayağa kaldırılması için Vagrant ve Virtualbox

3.7 REST Servisler

"Gerçek Zamanlı Karmaşık Olay İşleme" bileşeni tarafından üretilen uyarılar ve zenginleştirilmiş mesaj içerikleri "Anlık Durum" bileşeni tarafından bellekte tutulurken aynı zaman "Veri Depolama" bileşeni tarafından da dağıtık mimaride 2 yedekli olarak saklanır. Uyarı ve mesaj içerikleri JSON API formatında "REST Servisleri" bileşeni aracılığı ile dış dünyaya açılır. Uç kullanıcılar tarafından bu servislerin tüketilebilmesi için kullanıcı doğrulama ve yetkilendirme mekanizmasından geçmesi gerekir. Kullanıcı doğrulama ve yetkilendirme hizmetleri REST Servisleri bileşeni tarafından yerine getirilmiştir.

3.8 Sonuçlar ve Bulgular

Mesaj işleme altyapısının, oluşturulacak uyarılara yönelik hazır altyapılar sunması (kayan pencere), işlenen mesajlarda bulunulan noktanın saklanarak olası hizmet verilememesi durumlarında geriye dönük tekrar mesaj işleme olanağı sağlaması bu tarz sistemlerin tarafımızdan geliştirilmesine gerek bırakmamıştır.

Mesaj Kuyruklama altyapısı tarafından gelen mesajların belirlenen bir süre ile yedekli olarak saklaması (1 hafta), mesaj işleme altyapısındaki bir hatanın bu süre boyunca tolere edilmesini sağlamaktadır. Ek olarak bu altyapı, ayarlanabilir bir mesaj tüketme paralelliği sağladığı için, mesaj işleme altyapısına destek olmaktadır.

Mesaj kuyruğundaki kritik noktalardan birinin, konu bazında hangi sayıda paralel (partition/bölüm sayısı) olarak mesaj tüketeceğinin proje başlangıç aşamasında doğru olarak tahmin edilmesi olmuştur, çünkü daha sonradan bu değer değiştirilememektedir. Ek olarak seçilen mesaj kuyruğu altyapısı, mesajların sırasını sadece bulunulan paralelliğe (partition) göre garanti etmektedir. Buna göre kayan pencere uygulamasında, aracın yönünün hesaplanması sırasında sıralı gelmeyen mesajlar yüzünden aracın yönü

hesaplanmasına sorun olmuştur. İlgili sorun Veri Alım bileşeni tarafında aracın ID'sine göre sabit bir mesaj kuyruğu bölümüne kaydedilmesi yoluyla çözülmüştür.

Projede karşılaşılan diğer bir sorun, dağıtık çalışan sistemde bir sorun olması durumunda bunun ayıklanması (debug) aşamasıdır. Burada uygulamanın hangi makinada hata verdiğinin tespit edilmesinin zorluğu sebebiyle, buraya bağlanarak ayıklama aracının çalıştırılması kısımlarında zorluklarla karşılaşılmıştır. Bu noktadaki asıl sıkıntı, burada kullanılan açık kaynak aracın çok detaylı hatalar üretmemesi sebebiyle, sistemin detaylı bir izleme aracı ile devamlı olarak izlenmesinin şart olmasıdır. Aksi durumda sistemdeki sıkıntı zamanında tespit edilememektedir.

Büyük veri mimarilerinin genel olarak bir sorunu, hem makine sayısının hem de yazılım bileşen sayısının fazla olması sebebiyle, sistemleri genel olarak sistem yönetimi ve izlemenin zor olması olarak değerlendirilmektedir. İşletim sistemleri, yazılım altyapıları ve yazılımın ayrı olarak izlenmesi gerekmektedir. Bu sistemleri kuracak, yapılandıracak ve yönetecek yetişmiş insanlara ihtiyaç duyulmaktadır.

Bu çalışma kapsamında IoT cihazlardan veri toplanması, işlenmesi ve saklanmasına yönelik gerçek bir projedeki büyük veri altyapısı ve yazılım mimarisi deneyimleri paylaşılmıştır. Proje başlangıç aşamasında veri büyüklüğünün yüksek miktarda olacağının öngörülmesi sebebiyle büyük veri mesaj işleme ve saklama altyapıları tercih edilmiştir.

Büyük veri sistemlerinin kurulumu, yönetimi ve geliştirilmesi gibi durumlarda nispeten zor olması ve yetişmiş elemana ihtiyaç duyulmasına rağmen sunduğu hata toleransı, ölçeklenebilirlik ve performans kabiliyetleri sebebiyle bu tarz sistemler için en iyi seçenek olarak değerlendirilmektedir.

Kaynaklar

1. Yongheng Wang, Kening Cao: A Proactive Complex Event Processing Method for Large-Scale Transportation Internet of Things. International Journal of Distributed Sensor Networks, Volume 2014.
2. Elasticsearch Homepage, https://www.elastic.co/products/logstash, son erişim 2017/09/10.
3. Apache Kafka Homepage, http://kafka.apache.org/, son erişim 2017/05/11.
4. Apache Storm Homepage, http://storm.apache.org/, son erişim 2017/05/11.
5. Apache Lucene Homepage, http://lucene.apache.org/, son erişim 2017/05/11.
6. Docker Homepage, https://www.docker.com, son erişim 2017/05/11.