Burak Cem KARA

İSKENDERUN TEKNİK
ÜNİVERSİTESİ
MÜHENDİSLİK VE FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

YCSB PLATFORMU İLE YENİ NESİL BULUT VERİ DEPOLAMA SİSTEMLERİNİN KARŞILAŞTIRILMASI

Burak Cem KARA

ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ
ANABİLİM DALI

MAYIS 2019

ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

MAYIS 2019

YCSB PLATFORMU İLE YENİ NESİL BULUT VERİ DEPOLAMA SİSTEMLERİNİN KARŞILAŞTIRILMASI

Burak Cem KARA

YÜKSEK LİSANS
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

İSKENDERUN TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK VE FEN BİLİMLERİ ENSTİTÜSÜ

MAYIS 2019

Burak Cem KARA tarafından hazırlanan " YCSB PLATFORMU İLE YENİ NESİL BULUT VERİ DEPOLAMA SİSTEMLERİNİN KARŞILAŞTIRILMASI " adlı tez çalışması aşağıdaki jüri tarafından OY BİRLİĞİ ile İskenderun Teknik Üniversitesi Elektrik Elektronik Mühendisliği Anabilim Dalında YÜKSEK LİSANS TEZİ olarak kabul edilmiştir.

Danışman: Dr. Öğr. Üyesi Yaşar DAŞDEMİR

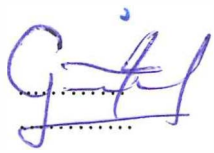
Bilgisayar Mühendisliği Anabilim Dalı, İskenderun Teknik Üniversitesi

Bu tezin, kapsam ve kalite olarak Yüksek Lisans Tezi olduğunu onaylıyorum.

Başkan: Prof. Dr. Yakup HAMEŞ

Elektrik Elektronik Mühendisliği Anabilim Dalı, İskenderun Teknik Üniversitesi

Bu tezin, kapsam ve kalite olarak Yüksek Lisans Tezi olduğunu onaylıyorum.



Üye: Dr. Öğr. Üyesi Erdem ASLAN

Elektrik Elektronik Mühendisliği Anabilim Dalı, Hatay Mustafa Kemal Üniversitesi

Bu tezin, kapsam ve kalite olarak Yüksek Lisans Tezi olduğunu onaylıyorum.

Tez Savunma Tarihi: 24/05/2019

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

Prof. Dr. Tolga DEPCİ

Mühendislik ve Fen Bilimleri Enstitüsü Müdürü

ETİK BEYAN

İskenderun Teknik Üniversitesi Mühendislik ve Fen Bilimleri Enstitüsü Tez Yazım Kurallarına uygun olarak hazırladığım bu tez çalışmasında;

- ☑ Yükseköğretim Kuruluna gönderilen kopya ile tarafından Mühendislik ve Fen Bilimleri Enstitüsü'ne verilen basılı ve/veya elektronik kopyaların birebir aynı olduğunu,
- ☑ Tez içinde sunduğum verileri, bilgileri ve dokümanları akademik ve etik kurallar çerçevesinde elde ettiğimi,
- ☑ Tüm bilgi, belge, değerlendirme ve sonuçları bilimsel etik ve ahlak kurallarına uygun olarak sunduğumu,
- ☑ Tez çalışmasında yararlandığım eserlerin tümüne uygun atıfta bulunarak kaynak gösterdiğimi,
- ☑ Kullanılan verilerde herhangi bir değişiklik yapmadığımı,
- ☑ Bu tezde sunduğum çalışmanın özgün olduğunu,

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

Burak Cem KARA

24/05/2019

YCSB PLATFORMU İLE YENİ NESİL BULUT VERİ DEPOLAMA SİSTEMLERİNİN KARŞILAŞTIRILMASI

(Yüksek Lisans Tezi)

Burak Cem KARA

İSKENDERUN TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK VE FEN BİLİMLERİ ENSTİTÜSÜ

Mayıs 2019

ÖZET

NoSQL veritabanı sistemleri gün geçtikçe daha hızlı bir şekilde büyük veri uygulamaları için yaygın olarak kullanılan bir veri platformu haline gelmektedir. Farklı NoSQL çözümleri arasından en iyi seçimi yapabilmek için belirli parametreler ışığında düşük veya yüksek veri yüküne sahip iş yükleri kullanılarak, bu sistemlerin zayıf ve güçlü yönlerinin analiz edilmesi gerekmektedir. Bu tezin amacı, üç farklı NoSQL veritabanı sistemlerini Windows işletim sistemi üzerinde çalıştırıp, düşük iş yükü altındaki yeteneklerini test ederek, her bir veritabanı sisteminin düşük iş yükü altındaki zayıf yönlerini ve güçlü yönlerini analiz etmektir. Çalışmamızda, NoSQL veri tabanlarının performanslarını test etmek için Yahoo'nun tasarladığı bir platform olan Yahoo Bulut Hizmet Ölçümünün (YCSB) default değerlerle sunduğu 6 farklı iş yüklerini kullandık. Ayrıca çalışmamızda MongoDB ve Cassandra veritabanı sistemlerinin güncel versiyonlarını kullanarak büyük verinin depolanması ve yönetimin de yaygın olarak kullanılan bu iki NoSQL sistemin son geliştirmelerle beraber kazandıkları yeni yeteneklerin performans analizlerini ortaya koyduk. Böylelikle Literatür araştırmalarımızda karşılaştığımız bu konuyla alakalı benzer çalışmalardaki MongoDB ve Cassandra gibi sistemlerin eski versiyonlarında yapılan analiz ve test sonuçlarını, çalışmamızdaki yeni versiyonlar da yapılan test sonuçlarıyla karşılaştırma imkânını sağladık. Bununla beraber popülerliği Çizge (Graph) veri tabanları dünyasında günden güne artan ve bu alanda en çok bilinen ve kullanılan Neo4J'ye alternatif olma yolunda ilerleyen OrientDB'yi MongoDB ve Cassandra gibi NoSQL veritabanları sistemleri dünyasının amiral gemisi konumundaki 2 sistemle karşılaştırarak, bu yeni veritabanı sisteminin NoSQL dünyasına kattıklarını gözler önüne serdik. Elde edilen sonuçlara göre, MongoDB ve OrientDB'nin düşük yükler altında saniye de gerçekleştirdiği iş miktarında Cassandra ya göre çok iyi performans gösterdiği, gecikme sürelerinde ise OrientDB'nin MongoDB'nin de üzerinde bir performans sergilediğini gözlemledik. Cassandra ise windows işletim platformu üzerindeki performansının bu iki veritabanı sistemine nazaran verimsiz kaldığını gözlemledik.

Anahtar Kelimeler : Büyük Veri, NoSQL, YCSB, MongoDB, Cassandra, OrientDB

Sayfa Adedi : 76

Danışman : Dr. Öğr. Üyesi Yaşar Daşdemir

COMPARISON OF NEW GENERATION CLOUD DATA STORAGE SYSTEMS WITH YCSB FRAMEWORK

(M. Sc. Thesis)

Burak Cem KARA

ISKENDERUN TECHNICAL UNIVERSITY
ENGINEERING AND SCIENCE INSTITUTE

Mayıs 2019

ABSTRACT

NoSQL database systems have been becoming commonly used data platform for big data applications. It is necessary to analyze this system's weak and strong sides by using workload with low and high pay load in the light of specific parameters to make best selection among different NoSQL solutions. The purpose of this thesis is to analyze that weakness and strengths of each database system's under low work load by launching three different NoSQL database systems on Windows operating system and testing their abilities under low work load. In our study, six different work loads of Yahoo Cloud Service Benchmark (YCSB), a platform designed by Yahoo were utilized in order to test performances of NoSQL databases. Moreover, widely used in storage and management of big data, these two NoSQL systems 'performances' analyses of new abilities which they gained with the recent developments have been demonstrated by using the current versions of the MongoDB and Cassandra database systems. Therefore, we have provided the opportunity to compare the analysis and test results of old versions of systems such as MongoDB and Cassandra in the similar studies we have encountered in our literature researches with the test results in the new versions of our study. Furthermore, contributions of OrientDB becoming more popular in graph database world and going ahead to be alternative to Neo4J used and known most in this area were revealed by comparing MongoDB and Cassandra two systems which are flagship of NoSQL database systems. According to result obtained, the fact that MongoDB and OrientDB showed better performance in throughput executed under low workload in second than Cassandra and that OrientDB showed better performance in latency than MongoDB were observed. Cassandra was insufficient in terms of performance on Windows operating system with regard to these two database systems.

Key Words : Big Data, NoSQL, YCSB, MongoDB, Cassandra, OrientDB
Page Number : 76
Supervisor : Assist. Prof. Dr. Yaşar Daşdemir

TEŞEKKÜR

Bu çalışmanın gerçekleştirilmesinde, değerli bilgilerini benimle paylaşan, kendisine ne zaman danışsam bana kıymetli zamanını ayırıp sabırla ve büyük bir ilgiyle bana faydalı olabilmek için elinden gelenden fazlasını sunan her sorun yaşadığımda yanına çekinmeden gidebildiğim, saygıdeğer danışman hocam Dr. Öğr. Üyesi Yaşar Daşdemir'e teşekkürü borç bilirim. Beni bu günlere sevgi ve saygı kelimelerinin anlamlarını bilecek şekilde yetiştirerek getiren ve benden hiçbir zaman desteğini esirgemeyen bu hayattaki en büyük şansım olan aileme sonsuz teşekkürler ederim.

İÇİNDEKİLER

Sayfa

Sayfa

ÇİZELGELERİN LİSTESİ

Çizelge **Sayfa**

ŞEKİLLERİN LİSTESİ

Şekil **Sayfa**

Sayfa

SİMGELER VE KISALTMALAR

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

Simgeler	**Açıklamalar**
ms	Milisaniye
opc/sec	Saniyedeki işlem miktarı

Kısaltmalar	**Açıklamalar**
API	Application Programming Interface
DBMS	Database Management System
ODBMS	Operational Database Management System
RDBMS	Relational Database Management System
P2P	Peer-to-Peer
REST	Representational State Transfer
YCSB	Yahoo! Cloud Serving Benchmark

1. GİRİŞ

Endüstri 4.0 ile birlikte günlük hayatımıza hızlı giriş yapan siber ekosistemindeki veri sirkülasyonu, büyük miktarda veri işlenmesinin zorluklarını beraberinde getirmiştir. Bu dönemde organizasyonların dijital ortamdaki bilgi ve verilere ulaşabilme ihtiyacı ve çabası her geçen gün katlanarak artmaktadır. Bu gelişim ve değişimler beraberinde bazı yapısal sorun ve karmaşaları da meydana getirmişlerdir. İnternet dünyasındaki birçok olgu 'Bilgi Çöplülügü' veya 'Bilgi Kirliliği' olarak anılmaya başlanmıştır. Bir düzene uymayan ve hacmi sürekli olarak artan ve büyüyen bu veri kirliliğinin anlamlı bir veri kümesine dönüşebileceğini düşünen ve inanan bazı yazılım firmaların çalışmaları sonucunda ortaya çıkan "Büyük Veri (Big Data)" kavramı aslında yıllardır biriktirdiğimiz verilerin anlamlı hale dönüştürülme çabasıdır. Google'ın ekonomi departmanın yöneticisi Hal Varian büyük veriyle ilgili olarak, bilgisayar teknolojilerinin kullanılmasıyla başlayan süreç ile 2003 arasında sadece beş exabyte veri oluşmuşken, günümüzde her iki günde bir bu tutara ulaştığımız bilgisini paylaşmıştır. 2020 yılına kadar olan beklenti ise, bu rakamın 53 zettabayt (53 trilyon gigabayt) miktarlarına gelmesidir. IBM ise, insanların artık günde 2,5 quintilyon byte veri oluşturduğunu söylemektedir.

Verinin büyüklüğü, miktarı ve karmaşıklığı gibi etkenlere bağlı olarak farklı veri modelleme, veri depolama ve sorgulama yöntemleri geliştirilmiştir [1]. Operasyonel veritabanları tarafından sağlanan en önemli hizmetlerden biri (veri depoları olarak da adlandırılır) kalıcılıktır. Kalıcılık, bir veritabanında depolanan verilerin izinsiz olarak değiştirilmeyeceğini ve işletme için önemli olduğu sürece kullanılabilir olacağını garanti eder. İçine koyduğunuz verileri korumak için güvenilir değilse, veritabanı ne işe yarar sorusunu kendimize sorabiliriz. Bu en önemli gereksinim göz önüne alındıktan sonra hangi veritabanı size daha iyi çözümler sunabilir soruna cevap bulabilmek adına, ne tür bir veri kullanacağını, veriye nasıl erişip güncelleyebileceğini ve hız mı yoksa veri güvenliğimi daha ön plan da olmalı konuların da düşünmeniz gerekmektedir. Bu en temel düzeydeki kurgular aslında, veritabanı yönetim sistemi seçmenizde ve verileri yönetmek ve işlemenizdeki genel başarınız için kritik öneme sahip işlemlerdir.

Dr. Edgar FrankCodd tarafından 1970 yılında yazılan "A Relational Model of Data for Large Shared Data Banks" isimli makalede ortaya çıkmış bir kavram olan ilişkisel veritabanları, tüm verilerin tablolar içerisinde yönetildiği, iki veya daha fazla tabloyu

birleştirilerek istenilen bilgilerin elde edilebildiği, verilerin doğru ve etkin bir biçimde saklanılması, veri bütünlüğü ve değişiklik kayıtları adlı yöntemleri sayesinde herhangi bir sorun yaşandığında verilerin hızlı bir şekilde kurtarıla bilinmesi gibi birçok özelliğe sahiptir. İlişkisel veritabanları sağladıkları yüksek verimlilik ve bütünlük ile beraber küçük veya büyük birçok şirket tarafından kullanılan bir sistem haline gelmiş ve bilişim dünyasında bir tabu olma yolunda ilerleyerek yıllarca veri depolama ve yönetiminde endüstri standardı haline gelmiştir.

Son yıllarda artan veri boyutları sebebiyle iyice kritik hale gelen veritabanı yönetim sistemleri alanında, artık ilişkisel veritabanları bu çapta büyüyen verileri tutmak ve bu verileri işlemek için yeterli değildir. İlişkisel veritabanların da, veri okuma, yazma ve güncelleme gibi işlemler Transaction işlemleri aracılığı ile yapabildiği için ve bu sayede tutarsızlık oluşmasına asla izin vermediğinden dolayı verinin birden fazla işlem tarafından değiştirilmesi engellenmiştir. İlişkisel veritabanları ile ancak gigabyte seviyelerinde veri saklana bilinir. Günümüzde veri boyutlarının ulaştığı bu noktada gigabyte seviyeleri çok küçük rakamlar olarak değerlendirilmektedir. Büyük verileri işleyebilecek veritabanı sistemleriyle ise petabyte seviyelerinde veriler saklana bilinmektedir.

Siber ortamdaki aygıtlar arasında yaşanan bu değişim ve gelişim, veri miktarlarını o kadar büyük boyutlara ulaştırır ki, ortaya çıkan bu büyük veri ilişkisel veri tabanı sistemlerinde tutulamaz bir hale gelir. Adlandırılamayan ve yönetilemeyen bu verileri bilgi çöplülüğü olmaktan kurtarmak ve organize edebilmek için NoSQL sistemlere başvurulmuştur [2]. Tasarımcısı ve fikir babası Carlo Strozzi tarafından 1998 yılında ortaya çıkarılan bu kavram, o güne kadar bilişim dünyasında vazgeçilmez olan ilişkisel veritabanlarına alternatif bir çözüm oldugunu vurgulamak ve teknoloji dünyasına etkileyici bir giriş yapabilmek adına, bu yeni sisteme ilişkisel olmayan anlamında No Relation ‘NoREL’ tarzında bir isim konulması gerektiğini söyler. NoSQL, internetin gün geçtikçe artan verisini depolayabilmek, büyük veriyi anlamlı ve yönetilebilinir bir yapıya kavuşturmak ve yüksek trafiğe sahip yapıların gereksinimlerini karşılayabilmek hedefiyle ortaya çıkmış yatay ölçeklendirme imkânına sahip dağıtık sistemlere verilen isimdir.

NoSQL, modern uygulamaların oluşturulmasında sunulan taleplere cevap olarak geliştirilen çok çeşitli farklı veritabanı teknolojilerini kapsamaktadır. Bir zamanlar sınırlı bir izleyici kitlesine hizmet veren bilgisayar uygulamaları, artık her zaman ulaşılabilir

olması gereken, birçok farklı cihazdan erişilebilen ve dünya çapında milyonlarca kullanıcıya ölçeklendirilmiş hizmetler olarak kullanılmaktadır. Kurumlar artık büyük sunucular ve depolama altyapısı yerine açık kaynaklı yazılım ve bulut bilişim kullanarak büyük ölçekli mimarilere yönelmektedir. Bilişim sektöründeki lider ve öncü şirketlerden Google'ın yıllarca indekslediği sitelerin bilgilerini, ilişkisel veritabanları gibi büyük verileri performanslı bir şekilde işleyemeyen pahalı sistemlerde tutmak yerine açık kaynaklı, esnek, ucuz ve performanslı sistem olan NoSQL veritabanı sistemi, Big Table üzerinde tutması bile NoSQL'in değerini ortaya koymaktadır.

Bu çalışmada NoSQL veritabanı sistemleri uygulamalarından Mongo DB, Cassandra ve OrientDB programları YCSB Workloads testlerine tabi tutulmuş olup, elde ettiğimiz performans sonuçlarının detaylı karşılaştırılması yapılmıştır. Bir verinin büyük veri olarak adlandırılabilmesi için sahip olması gereken 5 temel bileşen açıklanmıştır. Bununla beraber NoSQL veritabanı sistemlerinin olumlu ve olumsuz yönleri hakkında bilgiler verilmiştir. NoSQL sistemlerin temel taşı olan CAP teoremi, BASE teoremi ve Tutarlılık Modeli gibi üç farklı karakteristik özellikleri incelenmiştir. MongoDB, Cassandra ve OrientDB 'nin veri yapıları detaylı bir şekilde anlatılmış, üç veritabanın sahip oldukları veri yapıları ile ilişkisel veritabanı sistemlerinin sahip olduğu veri yapısı arasında karşılaştırmalar yapılmıştır. MongoDB. Cassandra ve OrientDB 'nin sahip oldukları mimari yapıları incelenmiş, bu üç sistemin ana özellikler hakkında detaylı açıklamalar yapılmıştır. Bölüm 5 de çalışmada kullanılan materyal yöntem ve kıyaslama aracına ait kod ve algoritmaları anlatılmıştır. Bölüm 6 da ise yapılan testlerin sonuçları grafikler ve tablolar halinde paylaşılmıştır.

2. BÜYÜK VERİ

Büyük veri (Big Data), geleneksel donanım ve yazılım sistemlerinin işleyemeyeceği (verinin işlenmesine cevap veremeyeceği) kadar çok miktardaki veri kümeleridir. Büyük veri terimi nispeten son yıllarda ortaya çıksa da, detaylı veri analizi için büyük miktarlarda bilgi toplama ve depolama eylemi uzun yıllardır kullanılmaktadır. Büyük verinin kullanılabilmesi belirli bir ölçeklendirmeye ihtiyaç duyulmaktadır. Bu bilgiler ışığında, ölçeklendirilemeyen veri, yönetimi zorlaştırmaktadır [2, 15].

Büyük verinin işlenerek kullanılması konusunda hem akademik çevreler hem de birçok teknoloji firması ciddi eforlar sarf etmektedirler. Büyük verinin türü ve içeriği ne olursa olsun verinin değerinin ortaya çıkartması için verinin toplanması, analizi ve görselleştirilmesi gibi üç aşamalı süreçten geçmesi gerekmektedir. Ayrıca elde edilen büyük miktardaki her veriyi büyük veri olarak adlandıramayız. Bir verinin büyük veri olup olmadığını belirleyen bir takım bileşenler mevcuttur. Bu bileşenleri Meta Group (şimdiki adı "GartnerGroup") analisti DougLaney, 2001 yılındaki bir araştırma raporunda büyük veriyi 3 boyutlu olarak 3V ile tanımlamıştır. Bu 3V: hacim (Volume), hız (Velocity) ve çeşitlilik (Variety).

Hacim (Volume)

Büyük verideki veri kümelerinin boyutlarını tanımlamakta kullanılmaktadır. Günümüzde teknolojinin geldiği son nokta itibariyle, veri miktarları terabayttan hatta petabayta kadar büyümektedir. Yalnızca sosyal ağ sitelerinden gelen Tweet'ler her gün 12 terabayt veriye tekabül etmektedir. 1.0x10100 değerine "Googol" denmektedir.

Hız (Velocity)

Hız, verinin elde edilme hızını göstermektedir. Örneğin Twitter üzerinden her dakika da atılan Twit'lerin gerçel zamanlı işlenmesi ve depolanması için hız önemli bir faktördür. Bu sebepten dolayı verilerin çok hızlı bir şekilde toplanması gerekmektedir. Bazen bir dakika gecikme bile analiz edilen çıktıda tutarsızlığa neden olabilir. Özellikle dolandırıcılık tespiti gibi zamana duyarlı sistemler de en iyi sonuçlar elde edebilmek için, oluşan büyük veriler, en hızlı şekilde analiz edilmelidir.

Çeşitlilik (Variety)

Çeşitlilik, büyük veri içerisindeki veri yapısı çeşitliliğini göstermektedir. Veriler, yapılandırılmış veya yapılandırılmamış niteliğinden bağımsız olarak herhangi bir türde olabilir. Veri türleri arasında metin, ses ve video verileri, sensör verileri, günlük dosyaları, e-posta mesajları vb. gibi birçok çeşitlilikte veri yapıları mevcuttur. İşletmelerin Büyük Veri ile çalışmayı tercih etmelerinin ana nedenlerinden biri, tüm bu veri türleri birlikte analiz edildiğinde yeni ve faydalı bilgiler elde edile bilinmektedir.

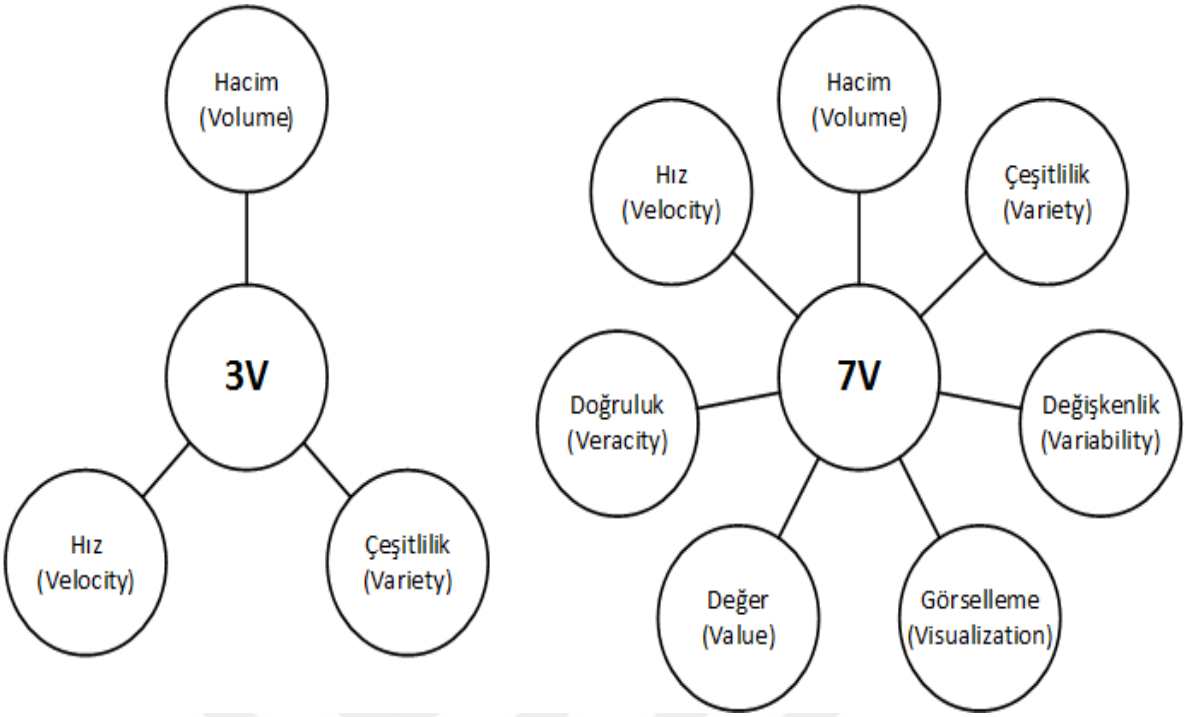
Daha sonraları bu "V" grupları 4V (3V'ye ek olarak Veracity), 5V(4V'ye ek olarak Value) [16], 7V (5V'ye ek olarak Variability, Visualization), 10V (7V'ye ek olarak Validity, Venue, Vocabulary, Vagueness) olarak farklı sayıda V özellikleri ile kullanılmıştır (Şekil 2.1).

Gerçeklik (Veracity)

Büyük Veri her geçen gün büyüklük ve çeşitlilikle büyüdükçe, ona güven duymak daha büyük zorluklar doğurmaktadır. Girdisine güvenilmeyen verinin, çıktısına da güvenilmez mantığında hareket ederek, iş dünyasındaki lider konumundaki firmalar, büyük veri den elde edilen bilgilere tamamen güvenmemektedirler. Ancak, daha iyi kararlar almak için bu bilgileri de göz ardı etmemektedirler.

Değer (Value)

Büyük verilerin anlamlandırılması yani değere dönüşmesidir. Yukarıdaki veri bileşenlerinden filtrelenen ve sonrasında analiz edilen verilerin, şirketlere artı değer sağlıyor olması gerekmektedir. Değere dönüşmüş verinin, alınacak kararlarda gerçek zamanlı olarak kullanılıp, karara anlık olarak etki edebilmeli ve doğru kararlar verebilmemiz de hemen elimizin altında olması gerekmektedir. Örneğin sağlık alanında önemli kararlar alan bir kamu kuruluşu çalışanları her an bölge, il, ilçe vb. detaylarda hastalık, ilaç, doktor dağılımlarını görebilmelidir.



Şekil 2.1. Büyük veriyi tanımlayan V grupları

3. NoSQL

Yıllar içerisinde bilgisayar ağlarının daha da büyüyerek karmaşıklaşması ve iletişim teknolojilerinin gelişimi, veri depolama sistemleri için bazı ihtiyaçlar ortaya çıkartmaktadır. Bu hızlı gelişim ile sensör ağları, sosyal medya ağları, dijital kütüphaneler ve arşivler, multimedya koleksiyonları ve web veri hizmetleri gibi kullanılan çeşitli bilgisayar sistemleri ürettikleri bilgiler ile ortaya büyük bir veri oluşturmaktadır [4]. Bu durum çok büyük verilerin üretildiği, bu verilerin saklanmasının, işlenmesinin ve yönetilmesinin çok önemli olduğu bir tasarıma ihtiyaç duymaktadır. Performans ve esneklik ihtiyacının giderek arttığı bu veri ortamında ilişkisel veritabanı yerine yeni bir veritabanı tasarımı ortaya çıkmaktadır. Bu yeni mimarinin ismi NoSQL olarak adlandırılmaktadır. NoSQL yani ilişkisel olmayan veritabanları bu açığı kapatmaktadır[2, 5]. Büyük Veri yapısını yöneterek ağ yapısındaki işleri kolaylaştırmaktadır.

Veritabanı sistemleri bilgisayar sistemi üzerinde veri tutarlılığını koruyarak sistemi maliyet ve hız açısından dengelerler [12]. Ancak ağ sistemimizde artan çok yönlü veri akışı ile ilişkisel veritabanları daha sınırlayıcı ve esneklikten uzak bir yapı içerisinde olduğu anlaşılmıştır. Bu açıdan, büyük verileri işleyen ve geniş ağa sahip veri bilimcileri büyük veri kavramında ihtiyaç duyulan yapıyı ilişkisel olmayan veri tabanları ile çözüme ulaştırmışlardır. İlişkisel veri tabanlarında bölünmezlik, tutarlılık, izolasyon ve dayanıklılık kavramları temel özelliklerdir [11]. Ancak çoklu yönlü veri akışı bu klasik özelliklerin geliştirilmesi ve yeni veritabanı ortaya çıkartılması ihtiyacı doğmuştur. Ayrıca büyük veri patlaması ilişkisel olmayan veri tabanlarının popülerliğinin arkasında asıl katalizördür. Geleneksel veritabanı yönetim sistemleri (DBMS'ler), önceden tanımlanmış şema ile yapılandırılmış veriler için tasarlanmıştır. Dolayısıyla, ilişkisel model (RDBMS), yaygın olarak büyük veri olarak bilinen yarı yapılandırılmış, yapılandırılmamış (Unstructured) veya diğer veri formlarıyla uğraşmayı çok zor bulmaktadır. Böylece verilerin ölçeklenememesi, esneklik dışı durumlar, düşük performans ve maliyet açısından eksiklikleri bulunan ilişkisel veri tabanlarının yerini NoSQL olarak adlandırılan ilişkisel olmayan veri tabanları almıştır.

3.1. NoSQL Sistemlerin Artı ve Eksi Yönleri

Yıllardır geliştirilen uygulamalarda verileri yönetmek ve işlemek için kullanılan Oracle, DB2, SQL Server, MySQL ve PostgreSQL gibi ilişkisel veritabanlarına alternatif olarak doğan bu yeni veritabanı sistemi, "NoSQL", yatay olarak genişletilebilir özelliği, esnek yapısı, açık kaynak ve ucuz maliyetlendirme gibi birçok artı yönleriyle Google, Facebook vb. gibi dev şirketlerin dikkatini çekmeyi başarmıştır. Neden NoSQL veritabanını kullanmalıyız soruna cevap olarak, NoSQL sistemlerin ilişkisel veritabanlarına göre artı yönlerini şu şekilde sıralayabiliriz;

- NoSQL sistemler yatay olarak genişletilebilirler. Bu sayede veriler farklı farklı depolama sistemlerinde tutulup, binlerce sunucu bir arada çalışarak bir küme (Cluster) meydana getirebilir ve bu küme (Cluster) içerisinde Büyük Veriler (Big Data) üzerinde kullanıcı veya geliştiriciler tarafından hedeflenen işlemler gerçekleştirilir. İlişkisel veritabanlarında ise bu durum sınırlıdır. Bu nedenle veri büyüdükçe daha pahalı bir yöntem olan dikey büyüme yaklaşımı tercih edilmektedir.

- NoSQL uygulamaları esnek yapıda olan sistemlerdir. Bu sebepten dolayı sistemin bakımı ve programlanması gibi işlemler kolaylıkla gerçekleştirilir. Ayrıca NoSQL veritabanı sistemlerinin birçoğu açık kaynak ve aynı zamanda bulut bilişimi (Cloud Computing) destekleyen sistemler oldukları için maliyet açısında da diğer veri tabanı sistemlerine oranla büyük avantajlara sahiplerdir.

- NoSQL veritabanları daha esnek bir veri modeline sahip olabilmek adına genellikle ilişkisel veritabanların da gördüğümüz bazı ACID özelliklerin den feragat ederek veya esneterek bu özelliklerden ödün verirler. Bu durum NoSQL veritabanları için olumsuz bir yön olarak gözükse de aslında, tek bir bulut sunucu performansının ulaşamayacağı derecede yatay ölçeklendirme gerektiren yüksek performanslı, düşük gecikme süreli kullanım durumlarında en iyi seçim olmasını sağlar.

NoSQL sistemler, bilişim dünyasına sundukları yeni çözüm yöntemleriyle ilişkisel veritabanlarına alternatif olma yolunda büyük aşamalar kaydetmişlerdir. Bununla beraber henüz ilişkisel veritabanları gibi 40 yıla yakın süredir sektörde kullanılan, bu sayede oluşan bir bilgi birikimi, kullanıcı kitlesi ve ürün olgunluğuna sahip olamadığı için bazı çevreler

tarafından halen tercih edilmemektedir. NoSQL sistemlerin ilişkisel veritabanlarına göre eksi yönlerini şu şekilde sıralayabiliriz;

- Günümüz de halen birçok firmanın ilişkisel veritabanlarını, NoSQL veritabanlarına tercih etmesindeki en önemli etmenlerin başında, NoSQL sistemlerin veri güvenliği konusunda İlişkisel veritabanları kadar gelişmiş özelliklere henüz sahip olmamasını gösterebiliriz. İlişkisel veritabanlarında gördüğümüz, veri kayıplarının yaşanmasının önüne geçen, birden fazla işlemin bir işlem olarak kabul edilmesi ve bu işlemlerden en az birinin olmaması durumunda hiç bir işlemin gerçekleştirilmediği bir yapı olan "Transaction" kavramı, NoSQL sistemlerde mevcut değildir. Bu sebeple NoSQL sistemler, veri güvenliğinin kritik derece de önemli olduğu konularda tercih edilmezler. Bu duruma bankacılık sektörünü örnek olarak gösterebiliriz.

- İlişkisel veritabanı sistemlerini kullanılarak geliştirilen bir uygulamanın NoSQL sistemlerine taşınması oldukça zahmetlidir. Çünkü ilişkisel ve NoSQL veritabanı sistemlerin tasarımları oldukça farklılık göstermektedir. Veriler başarılı bir şekilde taşınsa bile Join kullanılan sorgularda düzenlemeler yapılması gerekmektedir. Dolayısıyla ilişkisel veritabanı kullanan ve biraz ilerlemiş bir projenin NoSQL veritabanına taşınması büyük bir maliyet doğurmaktadır.

Genel olarak bakıldığında birbirinin alternatifi olarak görülen bu sistemler aslında birbirini tamamlayıcı özellikler de göstermektedir. Örneğin, ilişkisel veritabanlarının yapısı NoSQL sistemlere oranla daha sistematik ve güvenlidir ancak sistematikliğin getirdiği bir dezavantaj olarak ilişkisel veritabanı sistemleri, NoSQL veritabanlarından yavaştırlar. NoSQL veritabanlarının yapısı gereği dağıtık çalışmaya müsait olması, NoSQL sistemlerin, ilişkisel veritabanlarına kıyasla daha esnek ve hızlı olmasını sağlamaktadır. Fakat ilişkisel veritabanlarının dağıtık bir yapı yerine verilerin çok net bir şekilde sınıflandırılabildikleri ve ilişkiler kurulabildikleri bir mantık da hareket etmesinden dolayı, ilişkisel veritabanları ile raporlama konusunda NoSQL veritabanı sistemlere göre çok daha başarılı sonuçlar elde etmek mümkündür. Bu ve benzeri yukarda bahsettiğimiz bir çok sebepten ötürü bu iki farklı sistemi birbirine rakip olarak görmek yerine, yüksek performanslı, güvenli ve datalardan enformasyon (anlam kazanmış veri) elde edebilmek için bu farklı özellikteki veritabanları yönetim sistemlerini birbirleriyle entegre ve sahip

oldukları özellikler çerçevesinde optimum çözümler sunan yapılar kurmak yönünde çalışmalar yapmak gerekmektedir.

3.2. NoSQL'in Temel İlkeleri

CAP teoremi, BASE teoremi ve Tutarlılık modeli NoSQL sistemlerin temel taşlarını oluşturan 3 farklı karakteristik özelliğidir [17]. Bu bölüm de NoSQL sistemlerde asgari olarak bulunması gereken bu üç farklı karakteristik özellikler incelenmiştir.

3.2.1. CAP Teoremi

CAP, California Üniversitesi'nde bilgisayar bilimcisi olarak görev yapan Eric Brewer tarafından 2000 yılında ortaya koyulan ve bazı kesimlerce Brewer Teoremi olarak da bilinen bir teoremdir [18]. Eric Brewer'in bu teoremi ortaya koymasının sebebini şu şekilde açıklayabiliriz. Geleneksel veritabanlarında, veri büyüklüğü, veri akış hızı, çeşitliliği, istemci sayısı vs. arttıkça veritabanı performansı düşmeye başlar. Yıllardır sektöre egemen olmuş ilişkisel veritabanı sistemlerinin öncü firmaları bu durumu aşmak için veritabanı uygulamasının ayarlarında ve veritabanı tasarımında bazı değişiklikler yapmışlardır. Ortaya konulan bu geçici çözümler çoğunlukla tatmin edici olsa da, teknolojinin geldiği son nokta itibariyle veri miktarlarındaki inanılmaz artış sebebiyle ihtiyaçlar öyle bir noktaya varır ki, veritabanı uygulamasının ayarlarında değişiklik yapmak gibi geçici çözümler yerine altyapılarda köklü değişimlerin yapılması gerekir. Sistemlerimizin performans değerlerini kalıcı olarak artırabilmemiz için iki türlü seçeneğimiz vardır. Bu kalıcı çözümün yolu sisteme, dikey ölçeklendirme (Vertical Scale Out) veya yatay ölçeklendirme (Horizontal Scale Out) uygulamadan geçmektedir. Veritabanının dikeyde ölçeklenebilir olması için çok güçlü aynı zamanda pahalı bir donanım kullanılması gerekmektedir fakat bu işleminde bir sonu vardır. Çünkü tek bir bilgisayarda teknolojinin imkân verdiği ölçüde performans artırımı gerçekleştirilebilir. Bunun yanında veritabanının yatayda ölçeklenebilir olması ucuz ve çok sayıda makinenin aynı anda kullanılması anlamına gelir ki; bu da bize performans artışını ucuz ve makul yoldan elde etmemize imkân sağlar. Yapınıza yatay ölçeklendirme uygulandığında ise ortaya dağıtık sistemler çıkar. Dağıtık sistem, birden fazla bilgisayar veya sunucunun birbirleri arasında iletişim kurması ve ağdaki bütün sunucu veya bilgisayarların tek bir donanımmış gibi hareket eder şekilde bir ağ bütünü olarak çalışmasına denir. Bu noktadan

itibaren devreye Eric Brewer tarafından ortaya konulan CAP Teoremi girer. Dağıtık sistemlerdeki asgari kurallar CAP teoremi ile ifade edilmektedir. Teorem, dağıtık bir sistemin Consistency, Availability ve Partition Tolerance koşullarından aynı anda en fazla 2 tanesine sahip olabileceğini söyler [19]. CAP teoremi aşağıda bahsedeceğimiz özelliklerin ilk harfleriyle isimlendirilmiştir. Özellikler şunlardır;

Tutarlılık (Consistency)

Dağıtık sistemlerde bir veriye erişilmek istenildiğin de her koşulda ve her zaman en güncel kayda ulaşabilmesi gerekmektedir. Örneğin, bir sunucu üzerinden veritabanınızda değişiklik yaptınız. Yaptığınız bu değişikliği ağınızdaki diğer sunuculardan da sorguladığınız da, aynı sonuca ulaşabilmeniz gerekmektedir. Bu durum sizin yapınızın tutarlı olduğunun bir göstergesidir.

Erişilebilirlik (Availability)

Dağıtık sistemler üzerinde çalışan bir veritabanının, sistemdeki sunuculardan biri ya da birkaçı arızalı olsa bile yine de ulaşıla bilinir ve aynı zaman da kayıt ekleyip sorgulanabilir olmalıdır. Yani sistem hatalı veya başarılı tüm taleplere her koşul atında yanıt verir durumda olmalıdır.

Parça toleransı (Partition Tolerance)

Parça toleransı yani sunucular arasındaki bağlantıyı gösterene düğümler üzerindeki parçanın eksilmesine tolerans özelliği, ağınızdaki sunucular arasındaki bağlantı arızaya bile uğrasa sistemin çalışmaya devam etmesi gerektiğini ifade eder. Yani sunucular arasında bağlantıyı gösteren düğümlerden bazıları çalışmasa dahi veya düğümler arası ağda bir hasar olsa dahi işler aksamamalıdır. Tabi ki eksilmeden bahsederken makul seviyedeki bir durumdan bahsetmekteyiz. Yapıda tüm sistemi çalışmaz hale getirecek problemlerde de mevcut olmuş olabilir. Dolayısıyla bu durumu da göz ardı etmememiz gerekmektedir.

Eric Brewer bu teoremi ortaya koyarken bir dağıtık sistemin yukarıda bahsettiğimiz bu 3 özellikten en fazla iki tanesine sahip olabileceğinden bahsetmiştik ve dolasıyla dağıtık sistemi kullanmak isteyen teknolojilerin ortaya koyduğumuz bu üç özelliğin bir tanesinden

feragat etmesi gerekmektedir (Şekil 3.1). Bu durumda dağıtık sistem yapısında olan yüzlerce farklı NoSQL veritabanı teknolojileri arasından iş modelinize uygun olanını belirlemenizde bu teorem bir dereceye kadar kolaylık sağlar. CAP teoremindeki önemli bir ayrıntı olarak bir sistemin dağıtık bir sistem olarak tanımlayabilmemiz için bölünebilme tolerans özelliğini mutlaka karşılamalıdır aksi takdirde sadece CAP teoremindeki tutarlılık ve erişilebilirlik özelliklerini karşılayan sistemler dağıtık sistemler olarak adlandırılamaz.

Tutarlılık & Erişebilirlik (Consistency & Availability (CA))

Tutarlılık ve erişilebilirlik özelliklerini sahip fakat bölünebilme toleransı olmayan veritabanı sistemleri bu modele dâhildir. Verilerin doğruluğunun kesin olmasının istendiği durumlarda, yani veri tutarlılığının her şeyden daha fazla ön planda olduğu sistemler için ideal durumdur (Finansal işlemler, bankacılık işlemleri vb.). SQL, MySQL, PostgreSQL gibi bütün RDBMS'ler bu gruba dâhildir.

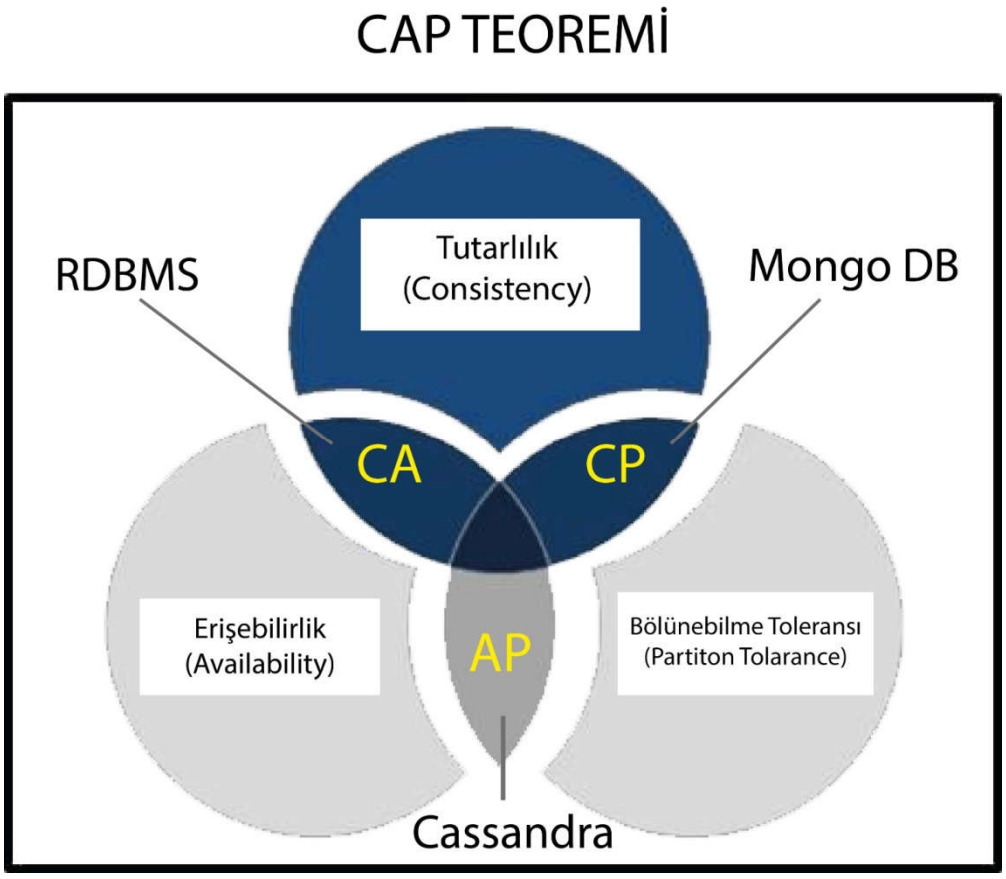
Tutarlılık & Hata Toleransı (Consistency & Partition Tolerance (CP))

Sunucular arası bağlantının bir bölümünde kopukluk oluşsa dahi sistemin çalışmaya devam ettiği ve tüm sunuculardaki aynı verinin görüldüğü mantıktaki yapılar bu modelleme türü altında toplanmaktadır. Erişilebilirlikten (Availability) feragat edilen kısım ise tutarlı olmayan verinin gösterilmemesi için bazı verilere erişim kısıtının konulmasıdır. Fakat erişilen kısımdaki verilerde tutarlılık korunmaktadır. Bu modelin uygulandığı sistemlere örnek olarak Hbase, Redis, MongoDB gibi NoSQL veri tabanı sistemlerini gösterebiliriz.

Erişilebilirlik & Hata Toleransı (Availability & Partition Tolerance (AP))

Güncel olmanın veya tutarlılığın ön planda olmasının gerekmediği sistemler için tercih edilen bir modeldir. Tutarlılıktan çok erişilebilirliğin ön planda olduğu iş modelleri için ideal bir durumdur. Örneğin sosyal medya da yapılan mesajlaşmaları bu modele örnek olarak verebiliriz. Bu model daha çok Facebook, Twitter vb. gibi akan verilerin olduğu platformlarda kullanılabilir. Bir kullanıcıya ait ‘Beğeni’ sayısının ne kadar olduğu bilgisinin herkes tarafından aynı şekilde görülmesinin ve bu sayının anlık olarak herkes için tutarlı olmasının bir önemi yoktur. Yani verilerde belirli bir süre için tutarsızlık

olabilir. Sonuç olarak bu modelde aranan özellik, sistemin sürekli olarak çalışmaya devam etmesidir. Tutarlılık ise ikinci plana atılmıştır.



Şekil 3.1. CAP Teoremi

3.2.2. BASE prensibi

BASE, basit olarak erişilebilirlik (BA), yumuşak durum (S), nihai tutarlılık (E)" , şeklinde tanımlanan ve ACID Teoremine taban tabana zıt olduğunu vurgulamak amacıyla bilerek bu şekilde isimlendirilmiş bir teoremdir [20]. NoSQL sistemler, performans ve erişilebilirlik yani sistemdeki sunuculardan biri ya da birkaçı arızalı olsa bile yine de ulaşıla bilinir ve aynı zaman da kayıt ekleyip sorgulanabilir olmalıdır yaklaşımının ön planda tutulduğu bir mantık da inşa edilmiş teknolojilerdir. Bu sebepten dolayı NoSQL sistemlerde, işlemlerin bir bütün olarak ele alındığı, işlemlerin tutarlı olması zorunluluğunun olduğu gibi birtakım sert kurallara sahip bir standart olan ACID teoreminin kullanılması mümkün değildir. ACID ilişkisel veritabanları sistemlerinde kullanılan bir tutarlılık modelidir. BASE ise, sistemde erişilebilirlik ilkesini ön planda tutulduğu, olmazsa olmaz bu en temel ilke sağlandıktan sonra tutarlılık ve eşzamanlılık

özelliklerinin de önemli olduğu veritabanı sistemlerinde tercih edilen bir yaklaşım modelidir. Kısacası ACID veri bütünlüğünün ve Tutarlılığının ön planda tutulduğu veritabanı sistemleri için uygun bir model iken BASE ise erişilebilirlik ilkesinin en temel öncelik olarak benimsendiği ayrıca daha kolay ölçeklenebilirliğin beklendiği veritabanı sistemlerinde kullanılan bir modeldir [7].

Basit olarak erişebilirlilik (Basically Available)

Bir veritabanı sisteminin her daim erişime açık durumda olması gerektiği prensibinin vurgulandığı yaklaşımdır. NoSQL sistemler verilere yüksek erişilebilirlik önceliğinin sağlanabilmesi için verilerini tek bir depolama alanı yerine Yineleme (Replication) özelliğine sahip birçok depolama alanında saklar ve böylece bir düğümde yaşanan aksaklık sistemin tamamına sirayet etmez ve sadece o verilerle kısıtlı kaldığı bir etki yaratarak tüm veri katmanın çalışır durumda kalması sağlanmış olur.

Yumuşak durum (Soft State)

BASE prensibini temel alan veritabanları sistemlerinin ACID modelindeki gibi katı bir tutarlılık gereksinime uymadığını ifade eder yani verileri yazılır ancak tutarlı olma zorunluluğu olmayabilir. BASE'in temel kavramlardan biri olan yumuşak durum (Soft State), veri tutarlılığının geliştiricinin problemi olduğunu ve veritabanı sistemlerinin tarafından ele alınmasının gerekli olmadığını savunmaktadır.

Nihai tutarlılık (Eventual Consistency)

NoSQL veritabanlarının tutarlılıkla ilkesine bakış açısı genel olarak şu yöndedir. Bir hatadan kaynaklı olarak veriler de zamana bağlı oluşan bir tutarsızlık durumunda, bu tutarsızlık geçici bir durumdur ve bir noktada itibaren bu veriler mutlak tutarlı bir hale dönüşecektir. Ancak sistem bu durumun ne zaman gerçekleşeceği konusunda hiçbir garanti vermemektedir. Bu durum BASE tabanlı yaklaşımları benimseyen sistemleri (NoSQL sistemler), bir işlemin önceki işlem tamamlanana ve veritabanının tutarlı bir duruma dönüştürülmesine kadar hiçbir işlemin gerçekleşmesine izin vermeyen ACID teoreminin derhal tutarlılık şartını benimseyen ilişkisel veritabanları sistemlerinden tamamen ayırmaktadır.

3.2.3. Tutarlılık modeli

Veritabanı teknolojilerini geliştiren firmalar sektörlerden gelen bu çağrıları yanıtsız bırakmayarak birçok farklı konu da değişik çözümler sunan yüzlerce yeni veritabanı teknolojileri geliştirmişlerdir. Veritabanı teknolojileri birçok konuda olduğu gibi sistemlerin gösterdikleri tutarlılık yaklaşımı için de temel olarak üç farklı yaklaşımı ele almışlardır [21]. Bu üç yaklaşımdan birinci durum, sistemlerin ilişkisel veri tabanı sistemlerinde gördüğümüz gibi son derece tutarlı olmasına dayanan güçlü tutarlılık modelidir. İkinci durum, veritabanı sisteminde yapılan bir güncellemenin daha sonra sunuculara yapılan erişimlerde yapılan bu güncelleme işleminin yapılan sorgu sonucunda kesin olarak aynı değeri getireceğini garanti etmeyen zayıf tutarlılık modelidir. Üçüncü durum ise sistemin tutarlılığı geri planda bırakarak, yapıda meydan gelen bir arıza olsa dahi işlemler kesintiye uğramadan devam etmeli ve sistem son derece hızlı çalışacak şekilde tasarlanmış olmalı yaklaşımına sahip olması mantığının ön planda olduğu nihai tutarlılık modelidir. Veri tutarlılığını en temel ilke olarak benimseyen veritabanlarında yazma işleminden hemen sonra veri sorgulandığında en güncel veriye anında ulaşılır. Veri tutarlılığının ikinci plan da olduğu veritabanların da ise yazma işlemi gerçekleştikten sonra bu işlem her sunucu tarafında hemen güncellenmemiş olabilir. Bunun sebebi ise dağıtık sistem yapısındaki veritabanı sistemlerinde verinin birden çok düğümde kopyalar halinde tutulmasıdır. Anlık olarak veri tutarlılığı sağlanmasa bile belirli bir süre sonrasında düğümlerin hepsinde kısa bir gecikmeyle dahi olsa da veriler eşitlenecektir. Fakat eşitlenme gerçekleşene kadar geçen sürede yapılan sorgulamalarda farklı sonuçlar dönecektir. NoSQL veritabanı sistemleri çoğunluğu veri tutarlılığını sağlamak amacıyla verileri Yineleme (Replication) olarak tanımladıkları yöntem ile farklı düğümlerde birkaç kopya halinde tutmaktadırlar. Her bir NoSQL veritabanı sistemi bu konuda farklı tutarlılık seviyelerinde çözümler sunmaktadır. Bazı NoSQL veritabanların da ise farklı veri tutarlılığı seviyeleri için bir takım yapılandırma seçenekleri de mevcuttur. HBase, BigTable gibi NoSQL veritabanları güçlü bir tutarlılık seviyesine sahip iken Cassandra ise verinin illaki bir noktadan itibaren tutarlı bir duruma dönüşeceği mantığında hareket eden nihai tutarlılık (Eventual Consistency) ilkesini benimsemiştir.

3.3. NoSQL Veritabanı Türleri

3.3.1. Geniş-sütun tabanlı

Verinin dinamik olarak sütun bazlı yapıda tutulduğu NoSQL veritabanları sistemine geniş-sütun tabanlı veri depolama denir. İlk NoSQL veri tabanı türü olan geniş-sütun tabanlı veritabanı sistemleri, ilişkisel veritabanlarının geleneksel tanım şemalarının aksine, verilerle çalışmak için önceden yapılandırılmış bir tablo gerektirmez. İlişkisel veri tabanlarında aşina olduğumuz şema yerine, sütun tabanlı depolama ortamları "Keyspace" olarak adlandırılan özelliği kullanırlar. Keyspace tabloya benzemekle beraber daha esnek bir formatta olup farklı sütunlar için birden fazla satırı ifade eden ve sütun gruplarını kuşatan bir mantık da hareket eder. Her bir kayıt aynı sayıda sütundan oluşmak zorunda değildir. Kayıtlar, bilgileri içeren bir veya daha fazla sütunla birlikte gelir ve her kaydın her sütunu farklı olabilir (Şekil 3.2). Temel olarak, sütun tabanlı NoSQL veritabanları, her bir anahtarın (yani satır / kayıt) kendisine bağlı bir veya daha fazla anahtar/değer çiftine sahip olduğu ve çok büyük ve yapılandırılmamış verilerin tutulmasına ve kullanılmasına izin verildiği (örn. tonlarca bilgi içeren bir kayıt) bir yapıda tasarlanmışlardır [22].

Sütun tabanlı veri tabanları, satır bazlı olanlara göre daha iyi bir sıkıştırma imkânı sunarlar. Ayrıca, büyük veri setleri kolay bir şekilde analiz edilir ve yorumlanır. Bu tür veri tabanları kümeleme türü sorguların ihtiyaç duyulduğu ortamlarda oldukça kullanışlıdır. Ancak olumsuz yönlerine baktığımızda veriyi yazma işlemi çok maliyetlidir ve ayrıca kayıtların toplu olarak güncellenmesi ve yüklenmesi kolay olmasına rağmen sadece belirli bir kayda yönelik güncelleme ve yükleme işlemi daha zordur. Ek olarak, geniş sütun tabanlı veri tabanları, ilişkisel veri tabanlarının üstün olduğu Transaction türü işlemlere uygun bir formata sahip değildir. Sütun tabanlı veri tabanları, yüksek okuma yazma performansı ve yüksek erişilebilirlik (High Availibity) için tasarlanmıştır. Dolayısıyla bu veritabanlarının kullanım alanları olarak hızın önemli olduğu büyük veri analizlerinde ve veri ambarı çözümlerinde tercih edilir. Transaction türü işlemlerin az olduğu büyük ölçekli projelerde kullanılır. Amazon DynamoDB, Apache Accumulo, Apache Cassandra, Apache HBase, Azure Tables, Bigtable, Hypertable, ScyllaDB ve Sqrrl NoSQL veritabanı sistemleri geniş-sütun tabanlı veritabanı ailesinin üyeleridir.

Satır tabanlı (RDBMS modeli)

id	İsim	Yaş	İlgi Alanları
1	Ahmet		Futbol, Film, Resim
2	Mehmet	25	
3	Hasan	30	Müzik

Geniş-sütun tabanlı modeli

id	İsim
1	Ahmet
2	Mehmet
3	Hasan

id	Yaş
2	25
3	30

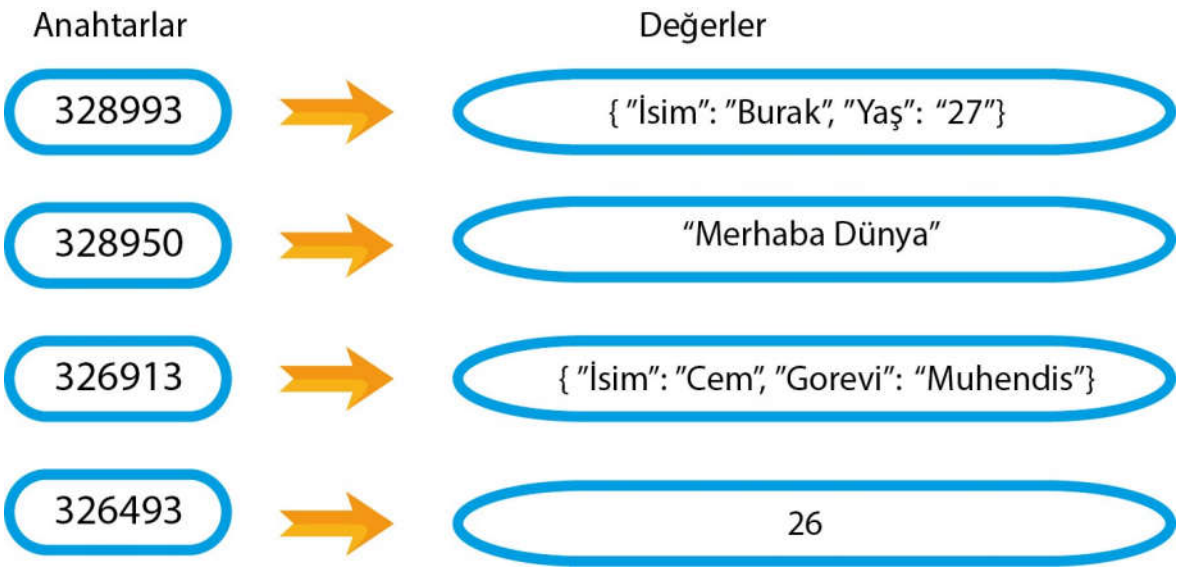
id	İlgi alanları
1	Futbol
1	Film
1	Resim
3	Müzik

Şekil 3.2. Geniş-sütun tabanlı modeli örneği [22]

3.3.2. Anahtar/değer tabanlı

Birçok kesim tarafından NoSQL veritabanı sistemlerinin türleri arasında en temel ve omurga türü olarak görülen, anahtar-değer depolama, verinin anahtar ve değer şeklinde tutulduğu depolama ortamıdır [23]. Sadece bir benzersiz anahtar (Key) karşılığında bir değer (Value) saklayabilen veri saklama desenine sahiptir (Şekil 3.3). Değer alanındaki veriye erişim için tek belirteç anahtardır. Anahtarlar veri tabanının izin verdiği her şey olabilir. Bu depolama türünce amaç anahtarlar üzerinden değerlere en hızlı şekilde ulaşabilmektir. Değerler ikili büyük nesneler veri tipi (Blob) olarak tutulur ve önceden tanımlı bir şemaya gereksinim yoktur. Dolayısıyla bu depolama türünde kolon kavramı yoktur. Bu tür veri tabanlarının birçok avantajları mevcuttur. Anahtar/değer ikilisi herhangi bir kod dönüşümü olmadan kolayca bir sistemden diğerine taşınabilir. Veriye erişmek için anahtar bilgisi kullanıldığından bu tür işlemlerde performans yüksektir. Anahtar/değer veritabanları diğer veritabanlarının ulaşamayacağı ölçeklerde yatay ölçeklendirmeye imkân tanır ve oldukça esnek olmakla birlikte her türde veriyi işleyebilir yapıdadırlar. Fakat bu kadar esnek yapıda olmanın da bazı olumsuz sonuçları vardır. Veriler ikili büyük nesneler veri tipi (Blob) olarak muhafaza edildiği için değerleri sorgulamak kolaydır fakat bu durum verinin raporlanmasını ve değerlerin belirli kısmının derlemesini zorlaştırmaktadır. Ayrıca her veriyi anahtar/değer ikilisi ile ifade etmek de mümkün değildir.

Anahtar/değer veritabanları sistemi, genellikle bir CPU ve hafızayı yoğun kullanan bir hesaplama yaptıktan sonra temel bilgilerin ve bazen de temel olmayan bilgilerin hızlı bir şekilde depolanması için kullanılır [24]. Son derece performanslıdırlar, verimlidirler ve genellikle kolayca ölçeklenebilirler. Sayı, metin, sayaç, JSON, XML, HTML, PHP, görüntü, video, liste. Vb.gibi her türlü veriyi, veri tipini belirtme zorunluluğu olmadan muhafaza edebilir. Azure Table Storage, Oracle Berkeley, Amazon WS Dynamo, Riak, Redis, Memcached NoSQL veritabanı sistemleri Anahtar/değer (Key/value) tabanlı veritabanı ailesinin üyeleridir.



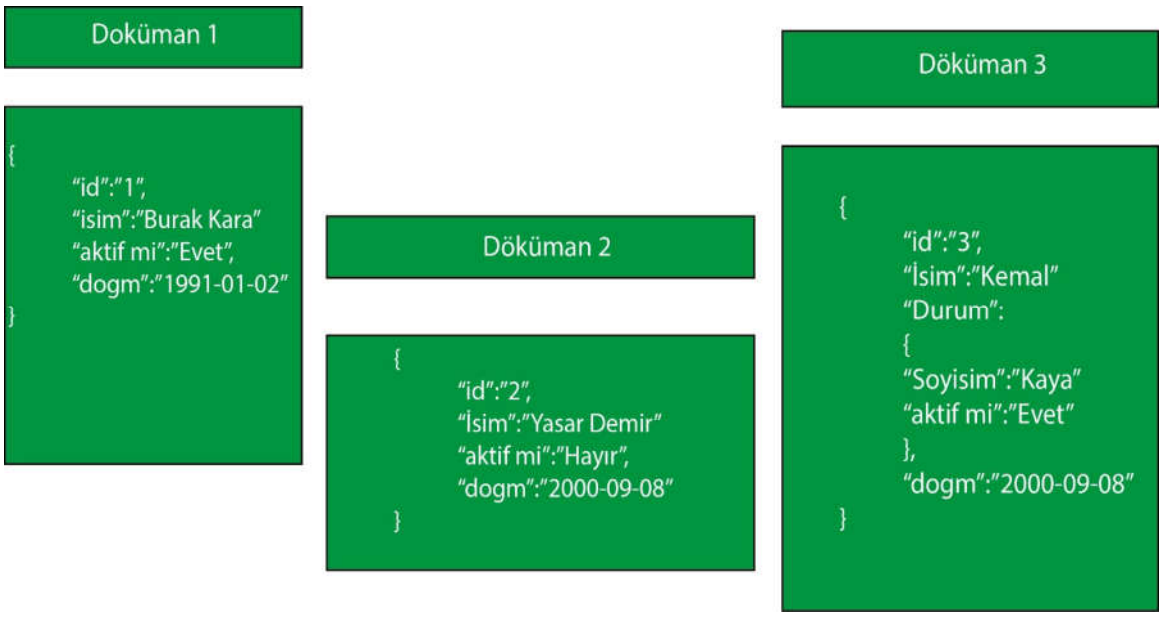
Şekil 3.3. Anahtar/değer tabanlı modeli örneği

3.3.3. Doküman tabanlı

Doküman tabanlı veritabanları, her belgenin benzersiz bir anahtarla tanımlandığı ve esnek şema olanağı sunan bir veritabanı sistemi geliştirme ihtiyacından ortaya çıkmış bir NoSQL veritabanı türüdür. Doküman veri tabanları veriyi JSON, BSON, veya XML doküman tipinde tutarlar. İlişkisel veritabanı sistemlerinde, veritabanına veriyi eklemeden önce şema tanımının yapılması gerekmektedir. Ancak doküman tabanlı veri tabanlarında önceden şema tanımına gerek olmadığı gibi doküman yapısını da belirtmeye gerek de yoktur. Dokümanlar istenen her tür veriyi içerebilir. Anahtar ve bu anahtara karşılık gelen değer ikililerine sahip olmakla birlikte üst veri (Metadata) özelliği ile veri kolayca sorgulanabilir [23]. Doküman tabanlı veritabanları, anahtar/değer veri tabanlarından farklı olarak veri

“Değer” kısmında, anahtar/değer olarak tutulur. Bu şekilde iç içe geçmiş yapı sayesinde bu dokümanların içerisinde sınırsız alan oluşturulabilir (Şekil 3.4).

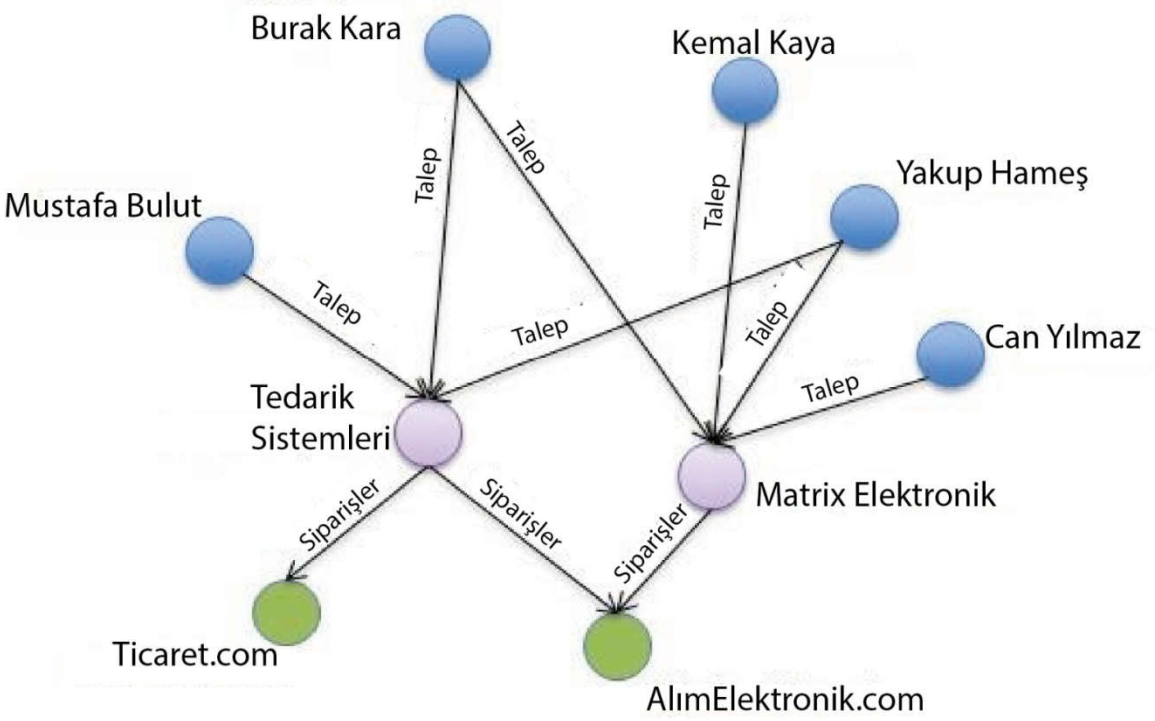
Doküman tabanlı sistemler, oldukça esnek bir yapıya sahiptir. Bu esneklik sistemin yatay olarak büyümesine izin verir. Bu NoSQL türüne ait veritabanı sistemlerinde, kullanıcılar diğer dokümanları etkilemeden istedikleri her türde yapıdaki veriyi yeni bir dokümanda saklayabilir, bu sayede yazma işlemleri hızlı gerçekleşir. Ayrıca hangi tür veriyi saklayacağınızı belirtmenize de gerek yoktur ve yapısal olmayan verilerde de oldukça iyilerdir. Dökuman tabanlı veritabanlarında, verilerin genellikle uygulama katmanında bir JSON belgesi olarak temsil edilmesinden dolayı, yazılım geliştiricileri için de en çok tercih edilen NoSQL veritabanı türüdür. Bunun sebebi yazılım geliştiricilerine veri modelini belge olarak düşünmek daha doğal gelmektedir. Doküman tabanlı veritabanlarının eksi yönleri olarak, Veri bütünlüğü ve tutarlılığının zayıf olmasını söyleyebiliriz. Bu veri tabanları, daha çok yapısal olmayan verilerin depolanmasında tercih edilmektedirler. Özellikle içerik yönetimi ve etraflı veri analizi yapılan uygulamalar için de uygundur. DynamoDB ve MongoDB gibi en popüler dökuman tabanlı NoSQL sistemler, esnek ve çevik yazılım geliştirme için güçlü ve sezgisel API'ler sağlayan popüler belge veritabanlarıdır [24].



Şekil 3.4. Doküman tabanlı modeli örneği

3.3.4. Çizge tabanlı

Çizge veritabanları, çizge kuramını kullanan bir tür NoSQL veri tabanıdır. Çizge Tabanlı NoSQL veritabanlarında, SQL'nin katı biçimini veya tablo ve sütun gösterimlerini bulamazsınız, bunun yerine ölçeklenebilirlik endişelerini gidermek için mükemmel olan esnek bir grafik gösterimi kullanılmıştır. Çizge veritabanları önceki üç NoSQL türünden tamamen farklı bir şekilde tasarlanmıştır. Ağaca benzer yapıları (yani grafikleri) ile veri depolamak için kenarları, özellikleri ve düğümleri (grafikteki varlıklar) olan grafik yapılarını (sonlu sıralı çiftler veya belirli varlıklar kümesi) kullanır (Şekil 3.5). Endeks içermeyen bitişiklik sağlar, yani her eleman doğrudan komşu elemanına bağlıdır ve dizin araması gerekmez. Veriler, Çizge Tabanı NoSQL veritabanı kullanılarak bir modelden diğerine kolayca dönüştürülebilir. Birleştirme işlemlerine ihtiyaç duymadıkları için de doğal olarak büyük veri kümelerine ölçeklenebilirler. Bir çizge veritabanının amacı, yüksek oranda bağlı veri kümeleriyle çalışan uygulamalar oluşturup çalıştırmayı kolaylaştırmaktır. Çizge veritabanları uygulamalarına örnek olarak Allegro, Neo4J, OrientDB, Virtuoso, Sesame veritabanlarını gösterebiliriz.



Şekil 3.5. Çizge tabanlı modeli örneği

3.4. NoSQL Veritabanı Yönetim Sistemleri

3.4.1. MongoDB

Bugün piyasada bulunan pek çok farklı yapıdaki NoSQL çözümlerinin arasında en popüler veritabanı sistemlerinin başında gelen MongoDB, 2007 yılında 10gen firmasın tarafından geliştirilmeye başlanmış ve 2009 yılında AGP lisansıyla açık kaynak projesine dönüştürülmüştür [25]. MongoDB, verilerini belge (Document) biçiminde saklayabilmek için kullanabileceğimiz, ölçeklenebilir (Scalable), C++ ile geliştirilmiş açık kaynak, NoSQL veritabanı uygulamasıdır [26]. MongoDB 'nin, GNU Affero General Public tarafından ücretsiz olarak lisanslanan bir sürümü de mevcuttur. Dil sürücüleri ise Apache tarafından lisanslanmıştır. Bununla birlikte 10gen firması, MongoDB için ticari lisanslar da sağlamaktır.

MongoDB'yi genel olarak, arşivleme ve loglama işleminin yapıldığı uygulamalar da, dinamik veri yapısına ihtiyaç duyulan sistemlerde ve büyük boyutlardaki verilerin okunması ve yazılmasının gerektiği uygulamalar da kullanmak daha uygun olabilir. Örnek verirsek 500 milyon kaydı olan bir veri kümesi düşünelim. Bu veri kümesinde son kullanıcı tarafından anlık sorgulama ihtiyacı olduğu durumlarda MongoDB gibi bir NoSQL veritabanı kullanabiliriz. Son yıllarda binlerce şirket, yeni uygulamalar geliştirmek, müşteri deneyimini iyileştirmek, hızlı pazarlama sürelerini kısaltmak ve maliyetleri en aza indirmek için MongoDB'yi kullanmaktadır [16]. eBay, Foursquare, Viacom, Craigslist ve New York Times gibi uluslararası boyutlarda şöhrete sahip birçok şirket MongoDB'yi kullanmaktadır.

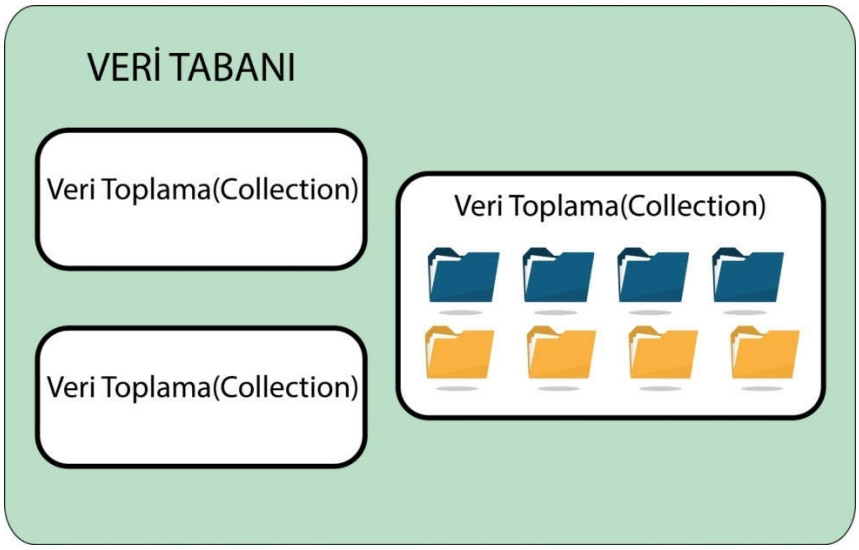
Veri modeli

MongoDB'de her kayıt, aslında bir dokümandır. Dokümanlar MongoDB'de JSON benzeri Binary JSON (BSON) formatında saklanır. BSON belgeleri, sakladıkları elemanların sıralı bir listesini içeren nesnelerdir. Her bir eleman, bir alan adı (Field) ve belirli tipte bir değerden (Value) oluşur (Şekil 3.6). İlişkisel veritabanlarında tablolar ile yapılan işlemler, MongoDB de koleksiyonlar(Collection) ile yapılır. Kısacası MongoDB'deki bir koleksiyon (Collection), ilişkisel veri tabanlarındaki bir tabloya karşılık gelir. MongoDB'deki bir

doküman, ilişkisel veri tabanlarında bir satırdır. MongoDB'deki bir alan ise ilişkisel veritabanlarındaki sütunlara karşılık gelmektedir.

Şekil 3.6. MongoDB doküman örneği

MongoDB, doküman odaklı bir veritabanıdır ve bu nedenle dokümanlar veri depolamak için birincil yapı olarak kullanılır [26]. MongoDB'nin veri modellemesi için kullandığı diğer kavramlar koleksiyonlar ve veritabanlarıdır. Bu kavramlar Şekil 3.7'de gösterilmektedir.



Şekil 3.7. MongoDB genel veri yapısı [27]

- Database, MongoDB konteksindeki, bir veritabanı derleme/yığın/koleksiyonlarından oluşan en dıştaki kapsayıcıdır [27].
- Koleksiyon(Collection), MongoDB belgelerinin bir grubudur.

- Belge (Document), tek bir veri birimini depolamak için en temel yapılardır. MongoDB belgeleri bir alan adından (Field: Value-Pairs) oluşan JSON benzeri Binary JSON (BSON) formatında saklar. Ek olarak, her belgenin benzersiz bir tanımlayıcısı olması gerekir. Temel veri türleri, diziler ve hatta nesneler, bir belgede saklanabilecek olası değerlerin örnekleridir [27].

Ana özellikleri

MongoDB'nin öne çıkan ve diğer NoSQL sistemlere oranla daha popüler ve tercih edilen sistem olmasında sorgu (Query) desteği, ikincil index desteği, Master-Slave Replication desteği, Sharding desteği, MapReduce desteği vs gibi özellikleri ön plana çıkmaktadır. Mongo DB'nin sahip olduğu temel özellikler şunlardır:

Dinamik şema (Dynamic schema)

MongoDB şema esnekliği sağlar. Başka bir deyişle, veri eklemeden önce şemayı tanımlamanıza gerek yoktur. Fakat yine de bir tasarıma muhtaçtır. Geliştiricilerin ve veritabanı yöneticilerinin ne tür sorguların işleneceğini, nesnelerin nasıl yönetildiğini ve belgelerin zaman içinde nasıl değiştiğini bilmeleri gerekir [28]. MongoDB, veritabanına ilişkin hayati ayrıntıları içeren bir tablo ve görünüm grubu olan merkezi sistem kataloğunu güncellemeden, sistemdeki diğer belgeleri etkilemeden ve sistemi çevrimdışına almak zorunda kalmadan bir belgeye yeni bir alanın eklenebileceği dinamik bir şema kullanır.

Gömülü veri modeli (Embedded data model)

MongoDB gömülü bir veri modeli kullanır. Başka bir deyişle, bir belgeyi başka bir belgede anahtar/değer çifti olarak tanımlayabiliriz. Gömülü veri modelleri desteği, veritabanı sistemindeki G / Ç etkinliğini azaltır [26].

Sorgu modeli (Query model)

MongoDB hemen hemen tüm büyük programlama dilleri için sürücülere sahip olduğu ve desteklediği için, MongoDB sorgu modeli, SQL'in yalnızca tek bir dil olarak kullanıldığı ilişkisel veritabanlarından farklı olarak bir programlama dilinin Uygulama Programlama Arabirimi (API),bir yazılımın başka bir yazılımda tanımlanmış işlevlerini kullanabilmesi

için oluşturulmuş bir tanım bütünü, içinde yöntemler veya işlevler olarak uygulanır. MongoDB'de çalıştırılabilecek sorgular:

- Anahtar/değer sorguları (Key/Value Queries) - Sonuçlar, belgedeki birincil anahtar gibi herhangi bir alana dayanmaktadır [28].
- Aralık sorguları (Range Queries) - Bu model de sonuçlar, eşit, eşit veya daha büyük, eşit veya daha küçük vs Gibi tanımlanan değerlerle elde edilir.
- Geospatial sorgular - Sonuçlar yan yana, kesişme veya çizgi, nokta, daire vb. Gibi ölçütlere dayanır.
- Metin arama (Text Search) - Sonuçlar, AND, OR, NOT gibi Boole işleçlerini kullanarak ilişki düzeylerine göre elde edilir. Boolean Operatörleri, bir aramada anahtar kelimeleri birleştirmek veya hariç tutmak için kullanılan, daha odaklanmış ve verimli sonuçlar veren basit kelimelerdir. (AND, OR, NOT veya AND NOT) [28].
- Toplama çerçevesi (Aggregation Framework) - Sonuçlar, min, max, ortalama gibi sorgularla döndürülen değerlere dayanır.
- MapReduce sorguları - Sonuçlar, veritabanında yürütülen karmaşık JavaScript sorgularına dayanmaktadır [22].

Dizin (Indexing)

MongoDB, bir sorgu işlendiğinde gereken disk erişimi sayısını en aza indirerek bir veritabanının performansını optimize etmeye yarayan veritabanı dizinlerini kullanmaktadır. Bu nedenle MongoDB, birçok dizin türünü desteklemektedir. İndeksleme sisteme ek bir yükte getirebilmektedir. Bu nedenle kullanılmayan indekslerin silinmesi veya indexlerin sisteme getirdiği fayda, zarar ilişkisi periyodik olarak incelenmelidir [22].

Sharding

MongoDB, donanım veya bulut altyapısı üzerindeki veritabanlarında, birden fazla düğüm (Node) üzerinden yatay ölçeklendirme yapılmasını sağlayan, Sharding adı verilen bir teknik kullanmaktadır. Sharding, çok büyük veritabanlarını veri parçaları adı verilen daha küçük, daha hızlı, daha kolay yönetilen parçalara ayıran bir tür veritabanı bölümlemesidir. Parça kelimesi bir veri bütünün küçük bir kısmı anlamına gelir. Bölünme (Sharding) ayrıca dağıtılmış depolama (Distributed Storage) olarak da yorumlanabilir. Dağıtılmış depolama

(Distributed Storage), bilgisayar sunucuları arasında bir ağ üzerinden paylaşılan dosyaların, verilerin ve veritabanlarının depolanma şekli olan merkezi depolamanın avantajları ile yerel depolamanın ölçeklenebilirliği ve düşük maliyet gibi özelliklerini birleştirme girişimidir. Sharding, veriyi Shards adı verilen birçok fiziksek parçalara ayırır ve bu sayede veritabanlarına donanım kaynaklı sınırlamaları aşmakta olanak sağlamaktadır [27].Bu durum Sharding özelliğine sahip MongoDB benzeri veritabanlarını, oldukça ölçeklenebilir hale getirdiği için onlara ilişkisel veritabanlarına kıyasla pozitif yönde önemli bir ayrım sağlamaktadır.

MapReduce

MapReduce, büyük veriyi büyük kümelerde (Clusters) işlemek üzere geliştirilmiş bir dağıtık programlama modelidir. MapReduce adından da anlaşılacağı üzere iki ana fonksiyondan oluşur. Map fonksiyonu, bir yığının tüm üyelerini sahip olduğu fonksiyon ile işleyip bir sonuç listesi döndürür. Reduce ise, paralel bir şekilde çalışan iki veya daha fazla Map fonksiyonundan dönen sonuçları harmanlar ve çözer. MongoDB, Paralel veri işlemek için Hadoop MapReduce'a benzer bir Map/Reduce çerçevesi kullanılarak gerçekleştirmektedir. MongoDB'nin MapReduce özelliği JavaScript ile kodlanmışken Hadoop'un MapReduce'u Java programla dili ile kodlanmıştır [26]. Genel olarak MapReduce modeli, İlişkisel veritabanlarındaki count, sum, having gibi işlemleri MongoDB üzerinde yapabilme için kullanılmaktadır.

Replication modeli

NoSQL teknolojilerinin zaman içinde gösterdikleri gelişimlerle beraber veri tabanlarında verileri tutma anlayışları da değişmektedir. Verilerin, küme içerisindeki sunucularda birden fazla kopyasının bulunması (Redundancy) ve bununla beraber yüksek erişilebilirlikten taviz vermemesi(High Availability), genel olarak her NoSQL sistem için olmazsa olmaz özellikler haline dönüşmüştür. NoSQL teknolojileri veri yineleme (Replication) işlemleri için farklı farklı çözümler sunmaktadırlar. MongoDB veri tabanı sisteminde veri kopyalama işlemleri Replica Set'ler aracılığıyla yapılmaktadır [26]. Replica Set, aynı veri setini içeren ve yöneten bir grup Mongodb servisleridir. Replica Set'lerin en önemli amacı aynı veri parçasını en az iki farklı yerde tutmak (Data Redundancy) ve bu verilerin yüksek erişilebilir (High Availability) olmasına imkân sağlamaktır. Replica Set'ler sayesinde,

küme içerisindeki bir sunucu (Node) da donanımsal bir problem olduğunda verilerin kopyası küme içerisindeki birden fazla sunucu (Node) da tutulduğu için, erişilemeyen sunucudaki veri, kopyasının tutulduğu sunucudan okunabilir. MongoDB sistemlerindeki Replica Set'ler ayrıca sistemin performansını arttırmak içinde kullanılır. Örneğin, veritabanından okuma işlemi yapılırken, verilerin kopyaları küme içerisindeki birden fazla sunucuda saklandığı için, veri okuma işlemlerinde tek bir sunucuya yüklenmek yerine, okuma yükü verinin kopyasının bulunduğu diğer sunuculara dağıtılarak, veri okuma hızı arttırılmış olur ve bu sayede sistemin de performansı artmış olur. Bu durum sadece veri okuma işlemleri için değil aynı şekilde veri yazma işlemleri içinde yapılır ve bu alan dada sistem de performans artışı sağlanmış olur.

3.4.2. Cassandra

NOSQL dünyasında ilk ve yaygın olarak kullanılan sistemlerinden biri olan Cassandra, büyük verileri yönetmek için kullanılan Apache Software Foundation tarafından desteklenen açık kaynaklı ve sütun tabanlı bir veritabanı yönetim sistemidir [29]. Geliştirilmesine ilk olarak Facebook tarafından başlanıp 2009 yılında Apache'ye devredilmiştir. NoSQL sistemleri için ortak sorgu işletim katmanı üzerinde çalışmalarda yapılmıştır [30].

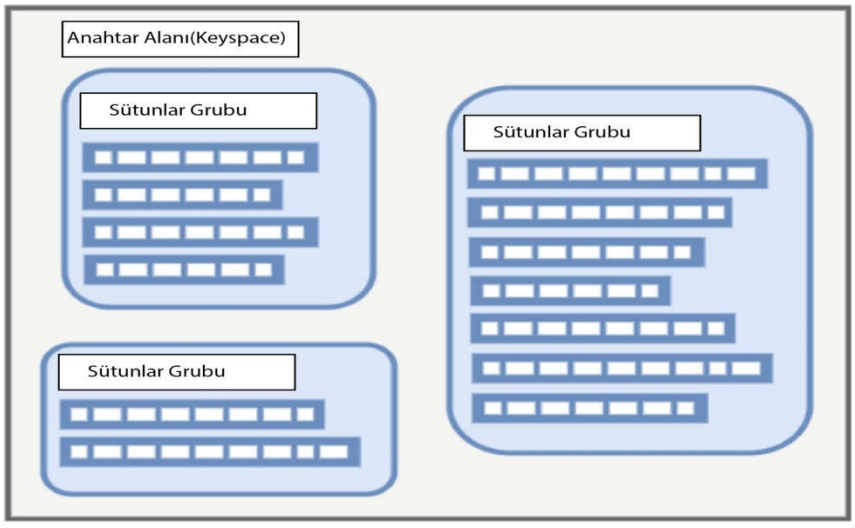
Facebook, dünyanın dört bir yanındaki veri merkezlerinde bulunan on binlerce sunucuyu kullanarak en yoğun zamanlarda yüz milyonlarca kullanıcıya hizmet veren en büyük sosyal ağ platformlarından biri olarak performans, güvenilirlik ve verimlilik gibi katı operasyonel gereksinimlere ihtiyaç duymaktadır. Bununla beraber platformun sürekli olarak büyümeyi destekleyen ve ölçeklenebilir bir yapıda olması da zaruri bir ihtiyaçtır. Her biri ufak bir yapıdan oluşan fakat binlerce miktarda sunucu ve ağ bileşenine sahip bir altyapıda belirli zaman aralıklarında arıza yaşanması olası bir durumdur. Bu nedenle, yazılım sistemlerinin bu tip durumları da göz ardı etmeden oluşturulmaları gerekmektedir. İşte yukarıda açıklanan tüm bu durumları, ihtiyaçları, gereksinimleri karşılamak için Facebook Cassandra'yı geliştirdi. Cassandra Java ile geliştirildiği için çalışırken JVM'e ihtiyaç duymaktadır [31]. Bundan dolayı önce Cassandra'nın çalışacağı işletim sistemi üzerine Java'nın kurulmuş olması gerekmektedir.

Cassandra, dağıtık bir NoSQL veritabanıdır. Dağıtık yapısı Amazon'un Dynamo, veri yapısı Google'ın BigTable veritabanları örnek alınarak geliştirilmiş olan Cassandra'nın yüksek ölçeklenebilirliğe ve yüksek performanslı bir dağıtık (Distributed) NoSQL mimarisine sahip olması dikkat çekmektedir. Cassandra, tek bir başarısızlık noktası olmayan (No Single Point of Failure) ve nihai tutarlılık (Eventual Consistency) özellikleriyle birlikte yüksek erişilebilirlik (High Availability) sağlamaktadır ve böylelikle BASE özelliklerine uyduğunu göstermektedir.

Cassandra'nın hedefi, kullanıcılara farklı veri merkezlerine yayılmış yüzlerce düğümden oluşan bir altyapı üzerinde çalışırken, yüksek erişilebilirlik sunan ve bunun yanında tek bir başarısızlık noktası olmayan (No Single Point of Failure) bir depolama sistemi çözümü sunmaktır. Cassandra'nın Kullanıcıları arasında Facebook, Twitter, Cisco, Rackspace, Digg, Cloudkick, Reddit, eBay, GitHub, GoDaddy, Instagram, Netflix gibi çok sayıda büyük firma bulunmaktadır.

Veri modeli

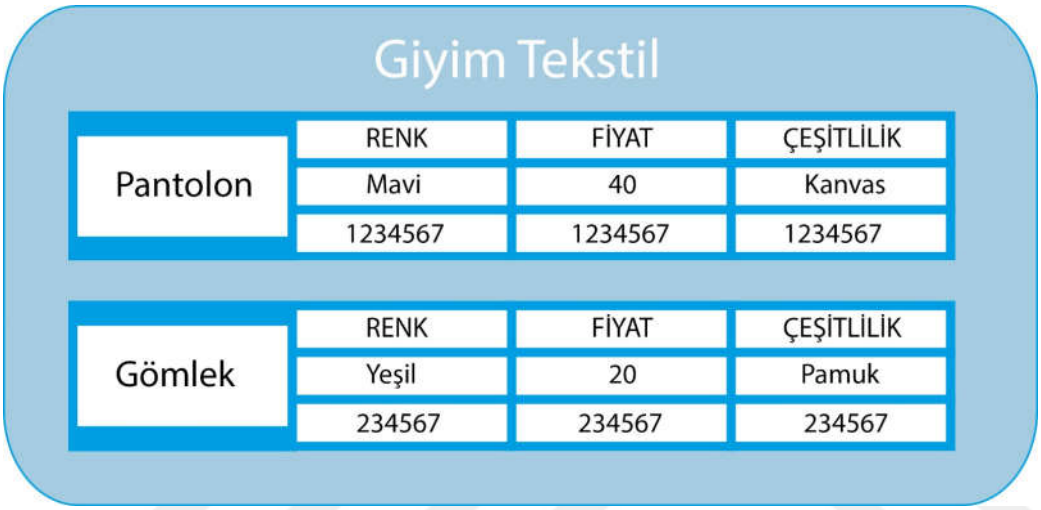
Cassandra da veri depolamak için kullanılan temel yapı sütunların kullanılmasına dayanır. Genel veri yapısı, bir ana alandan (Keyspace), sütun ailelerinden (Column Families), satırlardan (Rows) ve sütunlardan (Column) oluşur [32]. Cassandra veri modelinde aynı sütun ailesindeki satırlar farklı sayıda sütun ve farklı veriler içerebilirler. Bu durumu Şekil 3.8 de görebiliyoruz.



Şekil 3.8. Sütun ailelerinden (column families) oluşan bir ana alan örneği [27]

Ana alan (Keyspace), tüm veri kümesi için en dıştaki kapsayıcı kısımdır. En basit tanımıyla tüm veritabanına karşılık gelen kısım olarak adlandırılır. Ana alanlar (Keyspaces), sütun ailelerinden oluşur. (bkz. Şekil 3.8.)

Bir sütun ailesi (A Column Family), ilişkisel veritabanlarındaki tablo mantığına karşılık gelen kısımdır ve satırlardan oluşur [33] (Çizelge 3.1). Şekil 3.9'daki sütun ailesi örneğinde bazı kıyafetler hakkında çeşitli veriler içermektedir.



Şekil 3.9. Cassandra'nın veri modelinden bir sütun ailesi örneği [27]

Her satır (Row), bir varlığı temsil eder ve satırlar, sütunlardan oluşur [27]. Her sütun da, varlığın bir özelliğine veya niteliğine karşılık gelir. Sütunlar satır başlarında tanımlandığından dolayı, bu satırlar farklı isimlere, sütun sayısını değiştirme ve veri türü çeşitliliğine sahip olabilirler. Bu nedenle, her sütun sadece o satırla sınırlıdır ve ilişkisel veritabanlarında olduğu gibi tüm satırları kapsamaz. Her satır aynı zamanda bir satır anahtarı veya bölüm anahtarı olarak bilinen benzersiz bir tanımlayıcıya sahiptir.

Her sütun (Cloumn), Timestamp olarak adlandırılan bir zaman damgası ile birlikte bir ad-değer çifti içerirler. Zaman damgası Cassandra tarafından eklenir ve aynı verilerin birden fazla kopyası olduğunda tutarlılığı sağlamak için kullanılırlar [27].

Şekil 3.8 ve 3.9 incelediğimizde, sütun ailelerinin sahip olduğu sıralarının farklı büyüklük ve içerik sayısında olduğu, yani farklı sayıda sütun ve farklı veriler içerdiklerini görebilirsiniz. Bu durum bize, Cassandra veri modelinin aynı sütun ailesindeki satırların farklı sayıda sütunlara sahip olduğu geniş bir şemaya sahip olduğunu göstermektedir.

Çizelge 3.1. İlişkisel model ile Cassandra veri modellerinin karşılaştırılması

İlişkisel Model	**Cassandra**
Veritabanı (Database)	Ana Alan (Keyspace)
Tablo (Table)	Sütun ailesi (Column family)
Satır (Row)	Kayıt (Record)
Sütun (Cloumn)	Sütun (Cloumn)

Ana özellikler

Cassandra dağıtık veritabanı mimarisi üzerine kurulmuş bir sisteme sahiptir. Sistem, bir küme (Cluster) ve bu kümeleri oluşturan bir veya daha fazla düğümlerden (Node) oluşur [33]. Kümeler (Cluster) birbirine bağlı sunuculardan oluşurlar. Düğümler (Node) ise bir veritabanı sunucusudur. Dolayısıyla veritabanı birden fazla sunucuda saklanır. Düğümlerde (Node) saklanan verilerin kopyası başka Düğümlerde de saklanmaktadır. Cassandra'nın eşler arası (Peer-to-Peer) mimarisi, dağıtık bir yapıda olması (Distributed), yinelemeyi ve çoklu veri merkezi yinelemesini desteklemesi (Supports Replication and Multi Data Center Replication), ölçeklenebilirlik (Scalability), hata toleransı (Fault-Tolerant),ayarlanabilir tutarlılık (Tunable Consistency), MapReduce desteği gibi kullanıcılarına sunduğu birçok özellik vardır.

Eşler arası mimari (Peer-to-Peer Architecture)

Eşler arası mimari (P2P mimarisi), her bir iş istasyonunun veya düğümün aynı yetenek ve sorumluluklara sahip olduğu, yaygın olarak kullanılan bir bilgisayar ağ mimarisidir. Casandra, Master-Slave yapısında çalışan birçok NoSQL sisteminin aksine eşler arası mimari (P2P mimarisi) mimarisine sahiptir [22]. Master-slave mimarisinde bir ana düğüm (Node) vardır ve diğer düğümler bu ana düğüm ile iletişim kurarlar. Eşler arası bir mimaride ise, tüm düğümler (Node) birbirleriyle iletişim halindedir. Bu da kurulumu ve bakımı çok daha kolay olan bir yapı oluşturulmasına imkân sağlamaktadır. Cassandra veritabanı sisteminin kurulu olduğu yapıdaki tüm düğümler birbirleri ile iletişim

için Gossiper protokolünü kullanmaktadır. Gossiper, bir kümeyi oluşturan düğümlerin kendi aralarındaki iletişim kurmak için kullanılan bir eşler arası iletişim protokolüdür. Ayrıca Gossiper protokolünün sahip olduğu diğer bir iş yükü ise, sistemdeki her bir düğümün erişilemeyen düğümler de dâhil olmak üzere birbirlerinin o andaki durumdan haberdar olduklarını bildiğinden emin olmaktan sorumludur.

Ölçeklenebilirlilik (Scalability)

Cassandra yüksek ölçeklenebilir bir yapıya sahiptir. Bu sayede çok büyük miktarlarda veri saklanmasına ve saniyeler içerisinde çok sayıda kullanıcı odaklı operasyonlarının gerçekleştirilmesine imkân sağlamaktadır. Cassandra doğrusal ölçeklenebilirliğe sahiptir [31]. Örneğin n sayıda düğüm (Node) ile saniyede 500.000 işlem gerçekleştirirken düğüm (Node) sayısı arttırıldığında işlem kapasitesi de artmaktadır.

Veri dağıtımı (Data distribution)

Cassandra veritabanı yönetim sistemlerinde, veri otomatik olarak küme ya da Peer-to-Peer mimarisine sahip olduğu için "Ring" olarak adlandırılan yapının içindeki düğümlere dağıtılır. Bu sayede yazılımcıların ya da Cassandra veritabanı yönetim sistemi proje sorumlularının küme içindeki verinin dağıtılması ile ilgili herhangi bir ekstra işlem yapmalarına gerek yoktur. Veri şeffaf yapıdaki parçalar halinde düğümlere gönderilir. Bu parçalar düğümlere gönderilirken ya rastgele seçilerek ya da sıralı bir şekilde gönderebilirsiniz. Sistem varsayılan olarak rastgele yöntemini seçmiştir.

Replication modeli

İlişkisel veritabanı sistemlerinde ve bazı NoSQL veritabanlarının içindeki karmaşık yineleme (Replication) düzenlerinin aksine Cassandra'da konfigurasyon son derece kolaydır. Veritabanı yönetcileri son derece basit bir yolla ne kadar veri kopyası istediğini belirtir ve geri kalan işlemleri Cassandra veritabanı sistemi kendisi otomatik olarak yapar. Cassandra da kopyalama seçenekleri aynı zamanda verinin otomatik olarak farklı fiziksel yerlerde, çoklu veri merkezlerinde ve bulut platformların saklanmasına izin verir. Cassandra çoklu veri merkezlerinde ve bulut platformlarından veri kopyalama işlemlerinde önde gelen ve başarısı kabul görmüş bir NoSQL veritabanıdır [31].

Ayarlanabilir tutarlılık (Tunable consistency)

Cassandra veritabanı sistemi, küme içinde ayarlanabilir veri tutarlılığı sunmaktadırlar. Bunun anlamı bir yazılımcı veya veritabanı yöneticisi öncelikle sistemi için veri tutarlılığının ne derece önemli olduğuna karar verir. Bu kararı verdikten sonra bu doğrultuda, veritabanı sisteminin tutarlılık derecesini ilişkisel veritabanların da olduğu gibi son derece tutarlı şeklinde veyahut bazı NoSQL veritabanları sistemlerinde olduğu gibi nihai tutarlılık modelin olması için ayarlama yapabilinmektedir. Örneğin kümeler içindeki tüm düğümlerden cevap alarak veriyi oluştur ya da bir veya bir kaç düğümden cevap aldıktan sonra geri kalanları güncelle gibi bir kaç farklı tutarlılık senaryosu kullanılır. Ayrıca tutarlılık bir yazılımcı veya veritabanı yöneticisi tarafından her bir işlem (SELECT, INSERT, UPDATE ve DELETE vb. gibi) sırasında tutarlılık seçeneklerini ayarlayabilir.

CQL kullanımı

Cassandra Sorgu Dili (CQL), Cassandra veritabanıyla iletişim kurmak için kullanılan ana dildir. Cassandra Sorgu Dili (CQL), SQL'e benzer bir sorgu dilidir [22]. Fakat yine de Cassandra'nın yapısı, mimarisi ve çalışma mantığı ilişkisel veritabanı yönetim sistemlerinden daha farklı olduğu için, sorgu yazarken veya veritabanını dizayn ederken yaklaşımımızı buna göre yapmamız gerekmektedir. SQL ile CQL sorgu dilleri birbirlerine çok benzer bir yapıdadırlar fakat birbirlerinin aynısı olmadıklarını da unutmamız gerekmektedir. Bu durumu bir örnekle açıklamak gerekirse, ilişkisel veritabanı yönetim sistemlerin de bulunan fakat Cassandra bulunmayan 'FOREIGN KEY' kavramı ve 'JOIN' deyimini ele alalım. Cassandra bir NoSQL veritabanı olduğu için, tabloların birbirleri ile doğrudan ilişkisi bulunmamaktadır, eğer tablolar arasında bir ilişki kurulmak isteniyorsa bu kod yazılarak gerçekleştirilir. Bundan dolayı Cassandra da, ilişkisel veritabanı sistemlerinde bulunan ve iki ya da daha fazla tabloyu aynı anda sorgulayarak bir sonuç tablosu (Result Table) oluşturmaya yarayan JOIN deyimi bulunmamaktadır. Ayrıca JOIN ile beraber FOREIGN KEY kavramı da bulunmamaktadır. Cassandra veritabanı ile etkileşime geçmenin en temel yolu Cqlsh kabuğunu kullanmaktır. Cqlsh komutunu kullanarak anahtar alanlar, tablolar oluşturabilir. Bunun yanında tablolar ekleme ve sorgulama yapma gibi birçok işlemi de gerçekleştirebilirsiniz.

3.4.3. OrientDB

OrientDB 2010 yılında Luca Garulli tarafından olarak C++ dilinde yazılmış olup daha sonra java programlama dili ile yeniden kodlanan ve Çizge (Graph) veri tabanları dünyasında en çok bilinen NoSQL veritabanı sistemi olan Neo4J'den farklı olarak, açık kaynak bir NoSQL veritabanı sistemidir. OrientDB'nin, OrientDB Community ve Edition OrientDB Enterprise Edition olmak üzere iki farklı sürümü de mevcuttur. OrientDB şirketi, IBM ve Tokyo Institute of Technology tarafından yapılan bağımsız bir kıyaslama çalışmasında, OrientDB'nin tüm iş yükleri arasındaki grafik işlemlerinde Neo4j'den 10 kat daha hızlı olduğunu gösterdiği bilgisini paylaşmaktadır [34]. OrientDB'nin, Gremlin dilini desteklemesi, Multi-Master Replication özelliğine sahip olması, REST API, ACID ve en önemlisi de SQL desteğinin olması bizlere SQL veritabanı sistemleri dünyasında ciddi bir yeri olduğunu göstermektedir [35]. OrientDB'nin tercih edilen bir sistem haline dönüşmesinin bir diğer ana nedeni de dağıtık yapısı için Hazelcast'den destek almasıdır [36].

OrientDB, belge, anahtar/değer, nesne yönelimli ve çizge tabanlı modellerle birleştirip ölçeklendirilebilir ve yüksek performanslı bir operasyonel veritabanı sunan ilk çoklu model açık kaynaklı NoSQL veritabanı yönetim sistemidir. Farklı türdeki verilerin farklı formatlarda saklanmasının daha sağlıklı bir sistem oluşturacağını savunan çoklu model veritabanları, operasyonel karmaşıklığı azaltmak ve veri tutarlılığını korumak için birçok modeli birleştirerek çoklu veri modellerine olan ihtiyacı karşılamak için ortaya çıkmıştır. OrientDB'nin savunduğu bir diğer görüş ise her ne kadar grafik veritabanları son yıllarda popülerlik kazanmış olsa da, yeterince ileri gidemiyorlar ve NoSQL ürünlerinin birçoğu sadece ilişkisel veritabanı sistemleri üzerine kurulu uygulamaları ölçeklendirmek için kullanılmaktadır. İşte bu noktada OrientDB kendisi gibi gelişmiş ikinci nesil NoSQL sistemler olarak adlandırılan ve çağın gereksinimlerine daha uygun olarak geliştirilmiş çoklu model özelliğine sahip NoSQL ürünlerinin daha fazla işlevsellik ve esneklik sağlayarak ilişkisel veritabanlarının yerini alabileceğini savunmaktadır.

Veri modeli

OrientDB veritabanı motoru (Database Engine), verileri depolamak ve işlemek için Çizge (Graph), Doküman (Document), Anahtar/Değer (Key/Value), ve Nesne (Object) model

tiplerinde destek vermektedir. OrientDB dört modelin tüm özelliklerini çekirdekte birleştiren gerçek Çok Modelli veritabanı sistemidir [35]. Bunu anlamı OrientDB'nin bu dört modelin sahip olduğu yetenekleri veri tabanı motoru üzerinde sadece bir arayüz olarak kullanılan bir sistem yapısında değildir. Bu aynı zamanda OrientDB'yi diğer çoklu model veri tabanı sistemlerinden ayıran temel farktır. Çünkü genel olarak birçok NoSQL veri tabanı sistemi kendi sistemlerini çoklu model (Multi-Model) sistem olarak adlandırsa da gerçekte ana modellerinin dışında diğer modellere ait özellikleri taklit eden bir API ile ek bir katman da uygularlar. Sahip oldukları asıl model dışındaki diğer modellerle ilgili olarak hız ve ölçeklenebilirlik konusunda sınırlı bir performans kabiliyetine sahiplerdir.

Dökuman model (Document model)

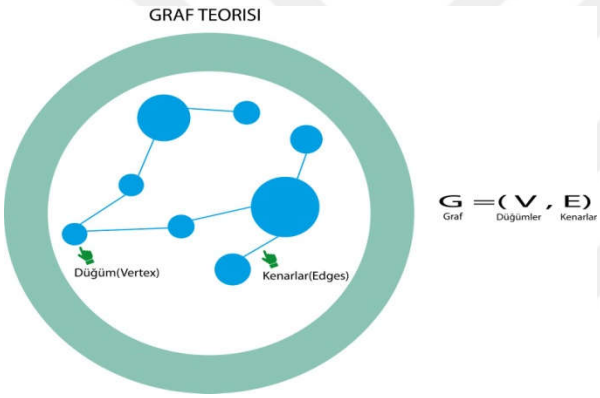
Dokuman tabanlı veritabanları, her belgenin benzersiz bir anahtarla tanımlandığı ve esnek şema olanağı sunan bir veritabanı sistemi geliştirme ihtiyacından ortaya çıkmış bir NoSQL veritabanı türüdür. OrientDB doküman modelinde belgeler koleksiyonlarda saklanır ve geliştiricilerin istedikleri tarzda verilerin gruplandırmasını sağlarlar. OrientDB, dokümanları gruplamak için dokuman tabanlı sistemlerde "koleksiyonlar" olarak adlandırılan kavramın yerine "sınıflar" ve "kümeler" kavramlarını kullanır (Çizelge 3.2). OrientDB'nin doküman modeli, doküman tabanlı NoSQL sistemlerde bulunmayan ve belgeler arasında bir ilişki olarak kullanılan "LINK" kavramını da ekler [35].

Çizelge 3.2. İlişkisel model, doküman modeli ve OrientDB doküman modellerinin karşılaştırılması

İlişkisel Model	**Doküman Modeli**	**OrientDB Doküman Modeli**
Tablo (Table)	Koleksiyonlar (Collection)	Sınıf veya Küme (Class or Cluster)
Satır (Row)	Doküman (Document)	Doküman (Document)
Sütun (Cloumn)	Anahtar / değer çifti (Key/value pair)	Doküman Alanı)Document field)
İlişki (Relationship)	Bulunmamaktadır	Bağlantı (Link)

Çizge model (Graph model)

Genel olarak, tablolar arası ilişkilerin yoğun olarak kullanıldığı veritabanlarını analiz etmek istediğinizde ya da hiyerarşik veri ile uğraştığınızda Graph Model'i kullanmanız işinizi kolaylaştıracaktır. Bir çizge (Graph) veritabanı, düğümler (Vertex), ilişki (Edge) ve özelliklerle beraber çizge (Graph)yapılarını kullanarak veriyi sunar ve saklar (Şekil 3.10). Bir çizge (Graph) modeli, İlişkiler (Edges) ile birbirine bağlanan düğümlerden (Vertices) oluşan ağa benzer bir yapıyı temsil eder. Düğümler (Vertex),insan, iş, hesap gibi takip edilmek istenen varlıkları (Entity) temsil ederler. İlişkiler (Edges) ise, düğümleri diğer düğümlere veya düğümleri özelliklere bağlayan çizgilerdir ve iki taraf arasındaki ilişkiyi temsil eder (Çizelge 3.3).



Şekil 3.10. Çizge (graph) modelindeki düğüm ve ilişkiler

Çizelge 3.3. İlişkisel model, çizge modeli ve OrientDB çizge modellerinin karşılaştırılması

İlişkisel Model	**Çizge Modeli**	**OrientDB Çizge Modeli**
Tablo (Table)	Düğüm ve İlişki sınıfları (Vertex and Edge Class)	Düğüm (Vertex) ve İlişkiye (Edge) uzayan(genişleyen) sınıflar
Satır (Row)	Düğüm (Vertex)	Düğüm (Vertex)
Sütun (Cloumn)	Düğüm ve İlişki Özellikleri (Vertex and Edge Property)	Düğüm ve İlişki Özellikleri (Vertex and Edge Property)
İlişki (Relationship)	İlişki (Edge)	İlişki (Edge)

Anahtar/değer model (Key/value model)

Anatar/Değer Modeli, OrientDB'nin kullanıcılara sunduğu diğer model tiplerine nazaran en basit yapıdaki türüdür. Bu model türünde veritabanındaki her veriye bir anahtarla (Key) ulaşılabilir ve değerler (Value), daha basit veya daha karmaşık yani detaylı bir şekilde tanımlanmaktadır. OrientDB'nin anahtar/değer modelinde, klasik bir anahtar/değer modelinde normalde bulabileceğinizden daha zengin bir değerler (Values) çeşitliliğini bulabilmekteyiz. Ayrıca OrientDB'nin anahtar/değer modeli, doküman ve çizge tabanlı modellerdeki değerlerinde (Values) kullanılmasına izin vericek desteğe de sahiplerdir. Klasik anahtar/değer modeli, anahtar/değer çiftlerini farklı "Container" da gruplamak için "Buckets" kullanırlar[35]. Aşağıdaki çizelge ilişkisel model, anahtar/değer modeli ve OrientDB anahtar/değer modeli arasındaki karşılaştırmayı göstermektedir (Çizelge 3.4).

Çizelge 3.4. İlişkisel model, anahtar/değer modeli ve OrientDB anahtar/değer modellerinin karşılaştırılması

İlişkisel Model	**Anahtar/Değer Modeli**	**OrientDB Anahtar/Değer Modeli**
Tablo (Table)	Kova (Bucket)	Sınıf veya Küme (Class or Cluster)
Satır (Row)	Anahtar / değer çifti (Key/value pair)	Doküman (Document)
Sütun (Cloumn)	Bulunmamaktadır	Doküman alanları veya Düğüm ve İlişki Özellikleri (Document field or Vertex and Edge Property)
İlişki (Relationship)	Bulunmamaktadır	Bağlantı (Link)

Nesne model (Object model)

Bir nesne veritabanı modeli, bilgilerin nesne yönelimli programlamada kullanılan nesneler biçiminde temsil edildiği bir veritabanı yönetim sistemidir. Nesne veritabanları, tablo yönelimli ilişkisel veritabanlarından farklıdır. Nesne-ilişkisel veritabanları ise her iki yaklaşımın bir karmasıdır. Nesne veritabanı modelleri, nesne yönelimli veritabanı yönetim

sistemleri (OODBMS'ler) olarak da adlandırılmaktadırlar ve veritabanı yeteneklerini nesne yönelimli programlama dili yetenekleriyle birleştirir. OODBMS'ler, nesne yönelimli programcıların ürünü geliştirmesine, onları nesne olarak saklamasına ve OODBMS içinde yeni nesneler oluşturmak için mevcut nesneleri çoğaltmasına veya değiştirmesine izin verir. Aşağıdaki çizelge, ilişkisel model, Nesne model ve OrientDB Nesne model arasındaki karşılaştırmayı göstermektedir (Çizelge 3.5).

Çizelge 3.5. İlişkisel model, nesne modeli ve OrientDB nesne modellerinin karşılaştırılması

İlişkisel Model	**Nesne Modeli**	**OrientDB Nesne Modeli**
Tablo (Table)	Sınıf (Class)	Sınıf veya Küme (Class or Cluster)
Satır (Row)	Nesne (Object)	Doküman (Document) veya Düğüm (Vertex)
Sütun (Cloumn)	Nesne özellikleri (Object property)	Doküman alanları veya Düğüm ve İlişki Özellikleri (Document field or Vertex and Edge Property)
İlişki (Relationship)	İşaretçi (Pointer)	Bağlantı (Link)

Ana özellikler

Replication

OrientDB, Multi Master Yineleme modelini (Multi-Master Replication) destekler. Bu model de, kümedeki tüm düğümler Master özelliğine sahiplerdir ve böylelikle küme içerisinde bulunan her bir düğüm veritabanın da okuma ve yazma yetkisine sahip olur. Küme içerisindeki herhangi bir sunucuda veya düğüm de yapılmış bir işlem, diğer tüm sunucular dada güncellenir ve böylelikle veritabanında tutarlılık sağlanmış olur. OrientDB, maksimum performans, ölçeklenebilirlik ve sağlamlık elde etmek için farklı sunucular arasında dağıtılmış, dağıtık sistemler olarak adlandırılan bir yapıda çalışır. Bu sayede sistem yatay olarak ölçeklendirebilir bir yapıya da kavuşmuş olur. OrientDB, düğümlerin otomatik keşfi, çalışma zamanı kümesi yapılandırmasını depolamak ve düğümler arasında

belirli operasyonları senkronize etmek için Hazelcast Open Source projesini kullanır. Hazelcast, açık kaynak kodlu Java tabanlı veri kümeleme, dağıtımı için yapılmış bir kütüphanedir [37]. Hazelcast verileri anlık olarak hafızada tutarak, Java Sanal Makinalar sayesinde verileri eşit şekilde balans etmeye çalışarak veri yükünü düzgün şekilde paylaştırıp, hızlı güvenli ve en az veri kaybı için çalışan bir yapıdır. Hazelcast ile java sanal makinalarınız ve sunucular arasındaki bu veri yükünün dağılımını belirleyebiliyoruz [37].

SQL desteği

Sorgu dilleri söz konusu olduğunda, SQL en çok tanınan standarttır. Geliştiricilerin çoğu, SQL konusunda tecrübeli ve daha rahat bir çalışma ortamına sahiplerdir. Bu nedenle Orient DB, sorgu dili olarak SQL kullanır ve sadece grafik işlevselliğini etkinleştirmek için bazı uzantılar eklemişlerdir. Standart SQL ile OrientDB tarafından desteklenen SQL arasında birkaç ufak farklılıklar vardır. Bu sayede ilişkisel veritabanlarında kurulu bir yapının OrientDB'ye taşınması konusunda büyük zorluklar yaşanmaz.

Gremlin desteği

OrientDB, Apache Tinkerpop'un grafik varlıkları oluşturmak ve grafik sorgu işlemlerini gerçekleştirmeye yönelik Gremlin API'si ve aynı zamanda bir grafik geçiş dili olan Gremlin'i destekler [35]. Grafik varlıkları (köşeler ve kenarlar) oluşturmak, bu varlıkların içindeki özellikleri değiştirmek, sorgu ve geçiş işlemleri gerçekleştirmek ve varlıkları silmek için Gremlin dilini kullanabilirsiniz.

ACID ve Rest api desteği

OrientDB, tüm veritabanı işlemlerinin güvenilir bir şekilde yapıldığının ve bir çökme durumunda bekleyen tüm belgelerin kurtarılıp işlendiğini garanti eden ACID işlemlerini destekler. ACID; Atomicity, Consistency, Isolation ve Durability kelimelerinin kısaltılmış halidir. REST istemci-sunucu arasında hızlı ve kolay şekilde iletişim kurulmasını sağlayan bir servis yapısıdır. REST, servis yönelimli mimari üzerine oluşturulan yazılımlarda kullanılan bir veri transfer yöntemidir. HTTP üzerinde çalışır ve diğer alternatiflere göre daha basittir, minimum içerikle veri alıp gönderdiği için de daha hızlıdır.

3.4.4. MongoDB, Cassandra ve OrientDB Karşılaştırma Tablosu

Aşağıdaki çizelge, Cassandra, MongoDB ve OrientDB'nin sahip oldukları genel özelliklerin karşılaştırmasını içermektedir (Çizelge 3.6).

Çizelge 3.6. MongoDB, Cassandra ve OrientDB NoSQL veritabanı sistemlerinin özellik karşılaştırılması

	Cassandra	**MongoDB**	**OrientDB**
Model	Kolon tabanlı (Wide column store)	Doküman tabanlı (Document store)	Grafik Tabanlı (Graph Database)
İnternet Sitesi	cassandra.apache.org	www.mongodb.com	orientdb.com
Teknik Dokümantasyon	cassandra.apache.org/-doc/latest	docs.mongodb.com/-manual	orientdb.org/docs/-3.0.x
Geliştiriciler	Apache Software Foundation	MongoDB, Inc	OrientDB LTD; CallidusCloud
İlk Sürüm Tarihi	2008	2009	2010
Lisans	Açık kaynak	Açık kaynak	Açık kaynak
Dil	Java	C++	Java
Sunucu işletim sistemleri	BSD Linux OS X Windows	Linux OS X Solaris Windows	Java JDK (>= JDK 6) uyumlu bütün işletim sistemleri
Tetikleyiciler (Triggers)	Var	Yok	Hooks
Bölümleme Yöntemleri (Partitioning methods)	Sharding	Sharding	Sharding
Çogaltma Yöntemleri (Replication methods)	Seçilebilir replikasyon faktörü	Master-slave replication	Master-master replication

Çizelge 3.6. (Devamı) MongoDB, Cassandra ve OrientDB NoSQL veritabanı sistemlerinin özellik karşılaştırılması

MapReduce Desteği	Var	Var	Yok
Tutarlılık Konsepti (Consistency)	Nihai Tutarlılık (Eventual Consistency) Hemen Tutarlılık (Immediate Consistency)	Nihai Tutarlılık (Eventual Consistency) Hemen Tutarlılık (Immediate Consistency)	Nihai Tutarlılık (Eventual Consistency) Hemen Tutarlılık (Immediate Consistency)
Foreign Keys	Yok	Yok	Var
Transaction Konsepti	Yok	Anlık görüntü yalıtımlı çok dokümanlı ACID Transactions	ACID
Eşzamanlılık (Concurrency)	Var	Var	Var
Dayanıklılık (Durability)	Var	Var	Var
Kulllanıcı Konsepti	Her nesne için kullanıcılara ayrı ayrı erişim izinleri tanımlanabilir.	Kullanıcılar ve roller için erişim izinleri tanımlanabilir.	Kullanıcılar ve rollller için erişim izinleri tanımlanabilir. Kayıt düzeyinde güvenlik yapılandırılmasıda mevcuttur.
Desteklediği Programlama Dilleri	C# C++ Clojure Erlang Go Haskell Java JavaScript Perl PHP Python Ruby Scala	C C# C++ Clojure Delphi Go Haskell Java JavaScript MatLab Perl PHP Python Ruby Scala	.Net C C# C++ Clojure Java JavaScript JavaScript (Node.js) PHP Python Ruby Scala

4. ÖNCEKİ ÇALIŞMALAR

Bilgisayar ve iletişim teknolojilerinde yaşanan hızlı gelişim, her geçen gün daha fazla organizasyonu etkileyerek farklı çözümler üretmeye zorlamaktadır. Günümüzde yaşanan bu değişim ve gelişim, verilerin modellenerek saklanmasını ve dolayısıyla veri tabanı kullanımını zorunlu kılmaktadır [1]. Veri tabanları, veri fazlalığını kontrol eden ve veri tutarlılığını koruyan en önemli sistemlerden biridir. [1]'deki çalışmada yazar ilişkisel ve ilişkisel olmayan veri tabanlarının karşılaştırmalı performans analizlerini yaparak sonuç ve öneriler sunmuştur.

Son yıllarda teknolojinin gelişmesiyle birlikte ortaya çıkan büyük veri kavramı depolama sistemlerinin altyapısının geliştirilmesi ihtiyacını ortaya çıkarmıştır [3]. Büyük verinin ortaya çıkış sebebi, temelde internet üzerinde boyutu her gün artan verinin, yüksek trafiğe sahip sistemlerin ihtiyacına cevap verebilecek hızda okunması/yazılması ihtiyacıdır. Bu ihtiyaç sonucunda karar verme ve bilgi keşif faaliyetlerini desteklemek için büyük miktarlarda veriyi etkin bir şekilde kullanmayı amaçlayan teknolojilerin ve yöntemlerin geliştirilmesi ve uygulanması gerekmektedir [4, 5, 6].

NoSQL veritabanları, geleneksel ilişkisel veritabanları tarafından ele alınamayan web tabanlı uygulamanın performans ve ölçeklenebilirlik gereksinimlerini karşılamak için tasarlanmıştır. İlişkisel veritabanları, ACID (Bölünmezlik, Tutarlılık, İzolasyon ve Dayanıklılık) özelliklerini güçlü bir şekilde takip ederken, NoSQL veritabanları BASE (Basically Available, Soft State, Eventual consistency) prensiplerini takip eder [7]. Bu çalışmada [7], Dynamo, Google Dosya Sistemi (GFS), Bigtable ve Hadoop gibi büyük ölçekli NoSQL'e odaklanan NoSQL veritabanının BASE özelliklerinin analitik çalışması yapılmıştır. NoSQL veritabanları dağıtık sistemlerdir ve yatay genişlemeye uygundur. Veri trafiği artması sonucu veriler tek bir noktada depolamak sistemi yavaşlatmaktadır. Hadoop benzeri yazılımlar ile birden fazla bilgisayarda saklayabiliriz ve yönetebiliriz [8, 9]. Böylece dağıtık veri tabanı oluşturularak hızlı veri akışı sağlanmış olur.

Bu büyük veri oluşumlarının ihtiyaçlarını karşılamak için ortaya çıkan NoSQL veri tabanı teknolojisi kavramsal modellemeyi analiz etmektedir. Son iki yılda, Twitter, Facebook, vb. gibi sosyal medyanın yaygın kullanımı bugün mevcut toplam verinin yaklaşık % 90'ını oluşturuyor ve bu büyük miktarda potansiyel olarak önemli veriler çeşitli dağıtılmış

yerlerde heterojen formatlarda saklanıyor [10]. Sugam Sharma ve arkadaşları [10]' deki makalesinde büyük veriyi bir petabyte büyüklüğünden öteye taşıyabildiğini iddia eden altı veri modelini ve sorgulama sistemlerinin mimarilerini test ederek, incelenen veri modellerinin yüksek kullanılabilirlik, yüksek ölçeklenebilirlik, sorgu dili vs. gibi özelliklerini karşılaştırıp, bir tabloda anlatmıştır.

Mongo DB, Cassandra, Hbase, Orient DB ve çok daha fazlası dâhil olmak üzere birçok NoSQL veri tabanları sistemleri çözümleri mevcuttur. Bu sistemlerden hangisinin sizin uygulamanız için uygun olduğuna karar vermek zor olabilir. Çünkü sistemler arasındaki özellikler farklılık gösterebilir ve ayrıca veritabanı sistemi kullanıcının, bir sistemin diğerine göre performansını karşılaştırarak hangisini kendisi için en uygun olduğuna karar vermesi kolay bir yol değildir.

Yahoo Cloud Serving Benchmark (YCSB) projesinin amacı, farklı mimari ve sisteme sahip olan bu veritabanlarının performanslarını kıyaslamak ve karşılaştırmak için bir çerçeve (Framework) ve ortak bir iş yükü seti geliştirmektir. Brian F. Cooper ve arkadaşları [11]'deki makalede Yahoo Cloud Serving Benchmark (YCSB) aracını kullanarak Cassandra, HBase, Yahoo! 'nun PNUTS ve basit bir MySQL uygulaması olan dört yaygın sistem için bir dizi karşılaştırma sonuçlarını ortaya koymuşlardır. Yazar, YCSB kıyaslama (Benchmark) aracını testlerinde kullanma sebebi olarak, iş yüklerin sistemlere kolay uyarlanması ve genişletilebilir bir yapıya sahip olması olarak anlatmaktadır.

Veri artışına bağlı olarak geliştirilen teknolojilerin ne ölçüde başarılı olduğu belirli analizler yapılarak değerlendirilmektedir. Hadoop teknolojisinin performansını değerlendirmek için geliştirilen HiBench teknolojisi detaylı analizlerde kullanılmaktadır. Colaso ve ark. [12]'deki çalışmasında detaylı bir benchmark çalışması yaparak NoSQL teknolojilerini karşılaştırmıştır. Cassandra, MongoDB, OrientDB ve Redis gibi dört modern NoSQL veritabanının simülasyona dayalı bir karakterizasyonunu gerçekleştirmişlerdir.

Günümüzde herhangi bir veritabanı yönetim sisteminin kullanılmadığı uygulama programlarına rastlamak mümkün değildir. Şirketlerin ihtiyaçları doğrultusunda birçok farklı amaca hizmet etmek için geliştirilen yüzlerce farklı veritabanı teknolojisnin

olduğunu söyleyebiliriz. Veritabanlarını iyi veya kötü diye ayırmak yerine ihtiyaç odaklı bir yaklaşımla ilerleyip, kullanıcının ihtiyaçları doğrultusunda en uygun olanı seçmek daha mantıklı bir yol olacaktır. Projene uygun veritabanını seçerken veri setleri ve veri modeli, işlem desteği, veri tutarlılığı ve performans açısından gereksinimlerini karşılayıp karşılamadığı belirlenmelidir. Christopher Jay Choi, tez bildirisinde [13] üç farklı NoSQL veritabanı sistemini, düşük ve yüksek veri yükleri altın da, YCSB Workload testlerine tabi tutmuş ve sonuçları tablolar ve grafikler halinde paylaşmıştır (Şekil 4.1).

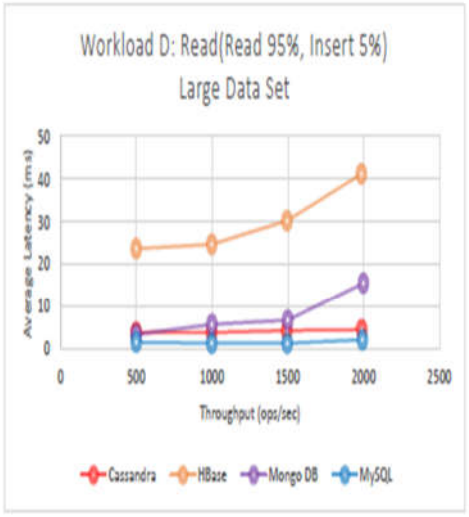


Figure 5.14 Workload D Read Large Data Set

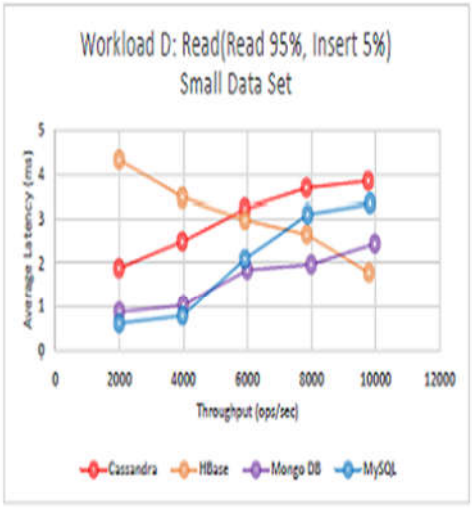


Figure 5.13 Workload D Read Small Data Set

Şekil 4.1. Christopher Jay Choi tezindeki Workload D sonuçları [13]

NoSQL sistemler sürekli olarak gelişmekte ve dolayısıyla kısa zaman dilimleri içerisinde Nosql ve Big Data kavramları üzerine yazılan makalelerin, yayınların ve araştırmaların sonuçları ve yazarların değerlendirmeleri geçerliliğini yitirmektedir. [14]' de 2015 yılında yapılan çalışmada, makalenin yazarları bazı Nosql Sistemlerin performanslarını test etmiştir ve aynı zamanda sürekli olarak gelişen ve yeni versiyonları yayınlanan NoSql veritabanları sistemleri ile ilgili geçmişte yapılan değerlendirmelerin eskimiş olduğunu vurgulayarak, o dönem için NoSQL sistemlerinin yeni ve güncel bir değerlendirmelerini ortaya koymuşlardır. Sonuç olarak incelenen tüm bu makalelerden yola çıkarak, çalışmam da sürekli olarak değişen ve gelişen NoSQL ve Big Data kavramlarına yeni bir bakış açısı katabilmek ve güncel Nosql veritabanı sistemlerinin performans analizlerini YCSB kıyaslama (Benchmark) aracı ile ortaya koyarak, bilişim dünyasının görüşüne sunulmuştur.

5. MATERYAL VE YÖNTEM

5.1. Materyal

Bu çalışmada NoSQL veritabanı sistemleri uygulamalarından MongoDB, Cassandra ve OrientDB programları kıyaslama aracı YCSB-0.12.0 sürümünün Workloads testlerine tabi tutulmuş olup, elde ettiğimiz performans sonuçlarının detaylı karşılaştırılması yapılmıştır. Bu çalışmada, 16 GB RAM, Intel Core i7-6700HQ CPU @ 2.60Hz (8 CPUs) bilgisayar konfigürasyonu kullanılmıştır. Depolama alanı olarak 128 GB 540 MB/s okuma, 130 MB/s yazma hızına sahip SSD disk kullanılmıştır.

5.1.1. Çalışmada kullanılan veritabanı sistemlerinin sürümleri

Her veritabanı sistemi için kendi web sitelerinde yayınlandıkları en son sürümler kullanılmıştır. Üç veritabanı sistemi de aynı işletim sistemi üzerine kurulmuştur. İşletim sitemi olarak Windows 10, 64 bit-lik sistem kullanılmıştır. Çalışmamda MongoDB web sitesinden indirdiğim MongoDB sürüm 3.6.3, Cassandra için Apache Cassandra web sitesinden indirdiğim Cassandra sürüm 3.11.2, OrientDB için ise OrientDB web sitesinden indirdiğim OrientDB sürüm 2.04 kullanılmıştır.

5.1.2. Çalışmada kullanılan kıyaslama aracı

Çalışmamızda Yahoo tarafından tasarlanan bir çerçeve (Framework) olan YCSB kıyaslama aracını kullandık. YCSB çeşitli tiplerdeki veritabanı sistemlerinin performansını ölçmek ve karşılaştırmak için kullanılan açık kaynaklı bir yazılım paketidir. YCSB projesinin amacı, farklı "anahtar/değer" ve "bulut" hizmet mağazalarının performansını değerlendirmek için bir çerçeve (Framework) ve ortak bir iş yükü seti geliştirmektir. Bu kıyaslama aracı ile MongoDB, Cassandra, Hbase ve çok daha fazlası dâhil olmak üzere birçok NoSQL veritabanı sistemlerinin performansını ölçmek ve karşılaştırmak mümkün olabilmektedir.

YCSB, günlük hayattaki ihtiyaçların kullanım durumlarını simüle etmek için çeşitli oranlarda CRUD (Oluşturma, Okuma, Güncelleme, Silme) işlemlerini gerçekleştirmek üzere tasarlanmıştır[39, 40]. Bu çalışmada, YCSB, düşük veri yükleri altında, gecikme

süresi ve saniyede yapılan işlem miktarlarını değerlendirerek bulut hizmet sistemlerinin performansını ölçmek için kullanılmıştır.

Workload İş yüklerini veritabanlarında çalıştırmak için 5 adım uygulanır [41].

1. Test edilecek veritabanı sisteme yüklenir.

2. Veri, test edilecek veritabanına yüklenir.

3. İş yükleri çalıştırılırken hangi parametreler kullanılacak ise bunlar belirlenir (threads, target, recordcount vs.).

4. Parametrelerde belirledikten sonra hangi iş yükünde veritabanı test edilecekse, o iş yükü belirlenir.

Seçilen iş yükü belirlenen parametrelerle birlikte çalıştırılır ve test sonuçları alınır.

5.2. Yöntem

Her bir veritabanına 1000000 kayıt yüklenmiştir. Bununla beraber her bir veritabanından 1000000 operasyon gerçekleştirilmesi istenmiş olup, bu operasyonları gerçekleştirirken 500, 1000, 1500, 2000, 5000 ve 10000'e ayarlanan hedef iş hacimleriyle, her bir veritabanından hedeflenen bu iş hacimlerini gerçekleştirilmesi istenmiştir.

5.2.1. İş yükleri (Workloads)

YCSB, okuma, yazma, güncelleme ve arama gibi farklı senaryoları birleştiren altı standart iş yüküyle birlikte gelir [15]. Her bir iş yükü (Workloads), sistemlerin performansları daha iyi değerlendirile bilinmesi açısından işlem sayılarına hedefler konularak ve toplamda 1000000 operasyonun gerçekleştirilmesi istenerek çalıştırılmıştır. Çizelge 5.1’de NoSQL sistemlerini karşılaştırırken kullandığımız iş yüklerinin, kullanım parametre oranları ve açıklamaları mevcuttur. Ayrıca her bir Workload sonucuna ait Average Latency hesaplarını yaparken de yine tabloda gösterilen oranlar kullanılmıştır. Average Latency değerlerini ulaşmak için örnek hesaplama formülü Workload A iş yükü için aşağıdaki gibidir.

Average (ortalama) Latency hesaplama Formülü;

∑((0,5*average(read)),(0,5*average(update)))

Çizelge 5.1. Kıyaslama (Benchmark) parametreleri

No	YCSB	Kıyaslama parametreleri
1	Workload A: Ağır iş yükü güncelleme	%50okuma ve %50 güncelleme. Bu iş yükü 50/50 oranında okunup yazılmıştır. Örneğin, en son eylemleri kaydeden bir kayıt günlükleri gibi.
2	Workload B: İş yükünü çoğunlukla okuma	%95 oku ve %5 güncelle. Bu iş yükü %95 okuma ve %5 yazma yapmaktadır. Örneğin; fotoğraf etiketleme ve etiket ekleme güncelleme olarak, etiketleri okumak ise okuma olarak kullanılır.
3	Workload C: Sadece okuma	%100 okuma işlemleri. Bu iş yükü %100 okuma yapar. Örneğin herhangi bir yerde üretilen kullanıcı profillerini önyükleme
4	Workload D: Son iş yükünü okuma	%95 okuma ve %5 ekleme. Bu iş yükünde yeni kayıtlar eklenir ve çoğunlukla eklenen kayıtlar kullanılmaktadır. Örneğin, kullanıcı durum güncellemeleri gibi.
5	Workload E: Kısa aralıklar	%95 tarama ve %5 ekleme. Bu iş yükünde bireysel kayıtlar yerine kısa kayıt aralıkları sorgulanır. Örneğin her bir taramanın verilen bir iş parçacığındaki yazıları taraması.
6	Workload F: Oku-değiştir-yaz	%50 oku ve %50 oku-değiştir-yaz. Bu iş yükünde istemci bir kayıt okuyacak, onu değiştirecek ve değişiklikleri tekrar geri yazacak. Örneğin; kullanıcı kayıtlarını okumak, değiştirmek ve kullanıcı aktivitelerini kaydetmek.

5.2.2. Kullanılan YCSB algoritması

İş yüklerini veri tabanlarına uygularken, kullanılan 6 iş yükü de benzer veri kümesine sahip olduğu için, veritabanı boyutunu tutarlı tutmak ve daha verimli ve düzgün karşılaştırmalar elde etmek için, çalışma sırasında aşağıdaki algoritma [41] kullanılmıştır:

1. Workload A parametre dosyasını kullanarak verileri, veritabanına yükle;

   ./bin/ycsbloadbasic -P workloads/workloada

2. Veriler, veritabanına yüklendikten sonra aşağıda belirtilen sırayla iş yüklerini çalıştır.

2.1. Workload A iş yükünü çalıştır.

./bin/ycsbrunbasic -P workloads/workloada

2.2. Workload B iş yükünü çalıştır.

./bin/ycsbrunbasic -P workloads/workloadb

2.3. Workload C iş yükünü çalıştır.

./bin/ycsbrunbasic -P workloads/workloadc

2.4. Workload F iş yükünü çalıştır.

./bin/ycsbrunbasic -P workloads/workloadf

2.5. Workload D iş yükünü çalıştır.

./bin/ycsbrunbasic -P workloads/workloadd

3. Workload D iş yükünü de çalıştırdıktan sonra Workload E iş yüküne geçmeden veritabanındaki veriler silinir.

4. Veritabanı yeniden başlatıldıktan sonra, Workload E parametre dosyası kullanılarak veriler, veritabanına yüklenir.

./bin/ycsbloadbasic -P workloads/workloade

5. Veriler, veritabanına yüklendikten sonra Workload E iş yükünü çalıştır.

./bin/ycsbrunbasic -P workloads/workloade

5.2.3. Kullanılan YCSB kodu

Bu çalışmada NoSQL veritabanı sistemlerine YCSB testlerini uygulaya bilmek için GitHub sitesinde yer alan kaynaklar kullanılmıştır. Aşağıda MongoDB için yapılan YCSB

testlerinden Workload A testi için Load Phase kodu paylaşılmıştır. Üç veri tabanı sistemi içinde hem Load Phase aşamasında hem de Transaction Phase aşamasında Workload testleri yapılırken benzer mantık da hareket edilmiştir.

Load Phase kodu

Aşağıdaki YCSB kodunda göreceğiniz üzere sisteme kodu girerken sadece gerçekleştirilmesi istenen iş yükü (Workload) yazılmamıştır. Yapılması istenen iş yüklerinin yanına belirli parametrelerde eklenmiştir.

ycsb load mongodb-async -p recordcount=1000000 -p operationcount=1000000 -p threadcount=100 -s -P D:\app\download\YCSB\workloads/workloada > A2outputLoad.txt

Yüklenen kod sonrasında sistemden beklenen ve gerçekleşmesi gereken cevap aşağıda paylaşılmıştır:

[OVERALL], RunTime(ms), 59124.0

[OVERALL], Throughput(ops/sec), 16913.605304106622

[TOTAL_GCS_PS_Scavenge], Count, 126.0

[TOTAL_GC_TIME_PS_Scavenge], Time(ms), 185.0

[TOTAL_GC_TIME_%_PS_Scavenge], Time(%), 0.31290169812597257

[TOTAL_GCS_PS_MarkSweep], Count, 0.0

[TOTAL_GC_TIME_PS_MarkSweep], Time(ms), 0.0

[TOTAL_GC_TIME_%_PS_MarkSweep], Time(%), 0.0

[TOTAL_GCs], Count, 126.0

[TOTAL_GC_TIME], Time(ms), 185.0

[TOTAL_GC_TIME_%], Time(%), 0.31290169812597257

[CLEANUP], Operations, 100.0

[CLEANUP], AverageLatency(us), 18.05

[CLEANUP], MinLatency(us), 1.0

[CLEANUP], MaxLatency(us), 1454.0

[CLEANUP], 95thPercentileLatency(us), 15.0

[CLEANUP], 99thPercentileLatency(us), 35.0

[INSERT], Operations, 1000000.0

[INSERT], AverageLatency(us), 5881.818859

[INSERT], MinLatency(us), 103.0

[INSERT], MaxLatency(us), 185215.0

[INSERT], 95thPercentileLatency(us), 13991.0

[INSERT], 99thPercentileLatency(us), 34079.0

[INSERT], Return=OK, 1000000

Yukarıda YCSB Load Phase evresi için girilen kod sonucunda dönen cevaba bakıldığından düzgün gerçekleşen bir işlem olduğu anlaşılmaktadır. Bu durumu MongoDB sisteminden gerçekleştirilmesi istenen 1000000 operasyonun tamamlandığı sonucundan anlıyoruz. Bizim çalışmamızda da sisteme girilen koda aynı şekilde cevaplar dönmelidir.

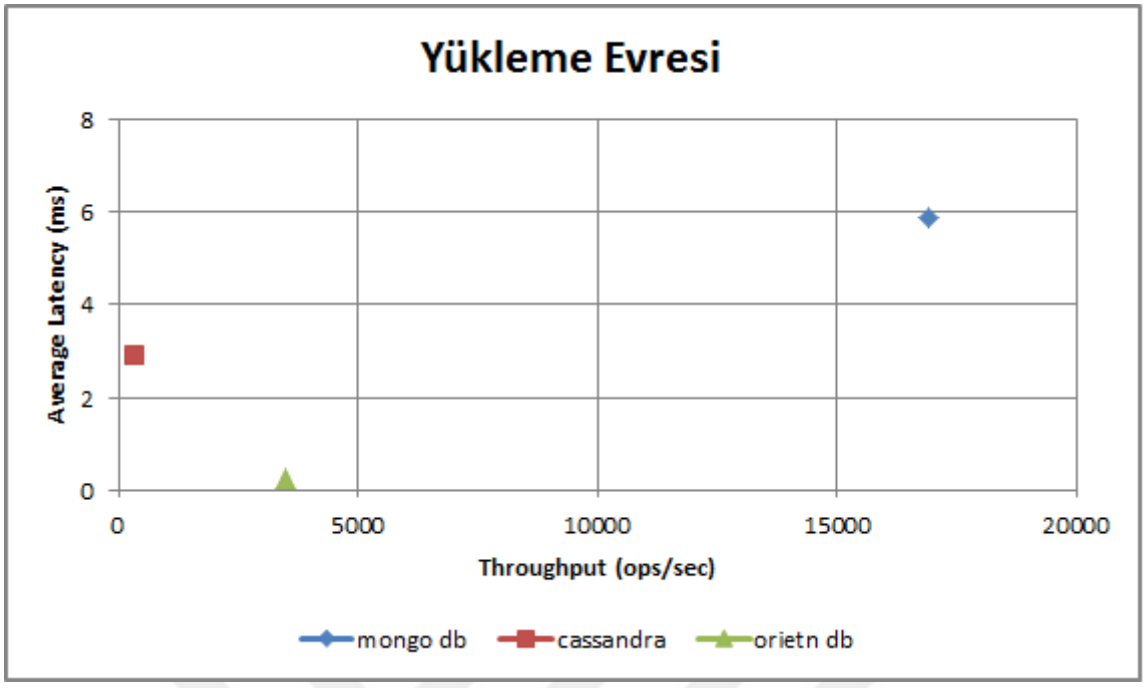
6. ARAŞTIRMA BULGULARI VE TARTIŞMA

6.1. Throughput ve Latency Sonuçları

Bu bölümde üç farklı NoSQL sistemine uyguladığımız iş yükleri testlerinden elde edilen saniyede gerçekleştirilen işlem miktarı yani Throughput ve her bir işlem arasındaki oluşan gecikme süreleri yani Latency sonuçları grafikler ve tablolar hainde paylaşılmıştır. Grafikteki Throughput ve Latency sonuçlarını değerlendirirken, Throughput için yüksek değere ulaşmış sistem için olumlu sonuçtur. Latency değerleri için ise düşük değerlere sahip veri tabanı sistemi için olumlu, yüksek değere sahip sistem için ise olumsuz sonuç elde dildiği anlaşılmalıdır.

6.1.1. Yükleme evresi

Yükleme evresinde eklenecek veriler tanımlanır. Altı iş yükünün hepsinin benzer bir veri kümesi olduğundan veri tabanı boyutunu tutarlı tutmak istediğimizden dolayı bu aşamada Workload A iş yükü veri tabanlarına yüklenmiştir. Daha sonra belirli bir sıralama düzeniyle her iş yükü için veri tabanı sistemleri teste tabi tutulmuştur. Çalışmamız da her bir veritabanının performansını gözlemleyebilmek için 1 milyon veri yükledik. Şekil 6.1'de eklediğimiz bu 1 milyon verinin karşısında veritabanı sistemlerinin gösterdiği performans değerlerini ortaya koyduk. En iyi yükleme süresine MongoDB ulaşırken, MongoDB'yi OrientDB ve ardından Cassandra takip etti, saniyede gerçekleştirilen işlem sayısında da yine MongoDB diğer iki veri tabanını sistemine oranla ciddi bir performans farkı ortaya koyduğunu grafik de gözlemliyoruz. Yine grafiği incelediğimiz de bu evrede en düşük ve en tutarlı gecikme sürelerini OrientDB'nin yakaladığını gözlemliyoruz, bununda yanında MongoDB'nin ise bu alanda en kötü performansı ortaya koyduğunu gözlemliyoruz. Çizelge 6.1'de Load Phase grafiğine ait sonuçlar detaylı olarak verilmiştir.



Şekil 6.1. Yükleme evresi (1 milyon kayıt)

Çizelge 6.1 Yükleme evresi Throughput ve Average Latency sonuçları

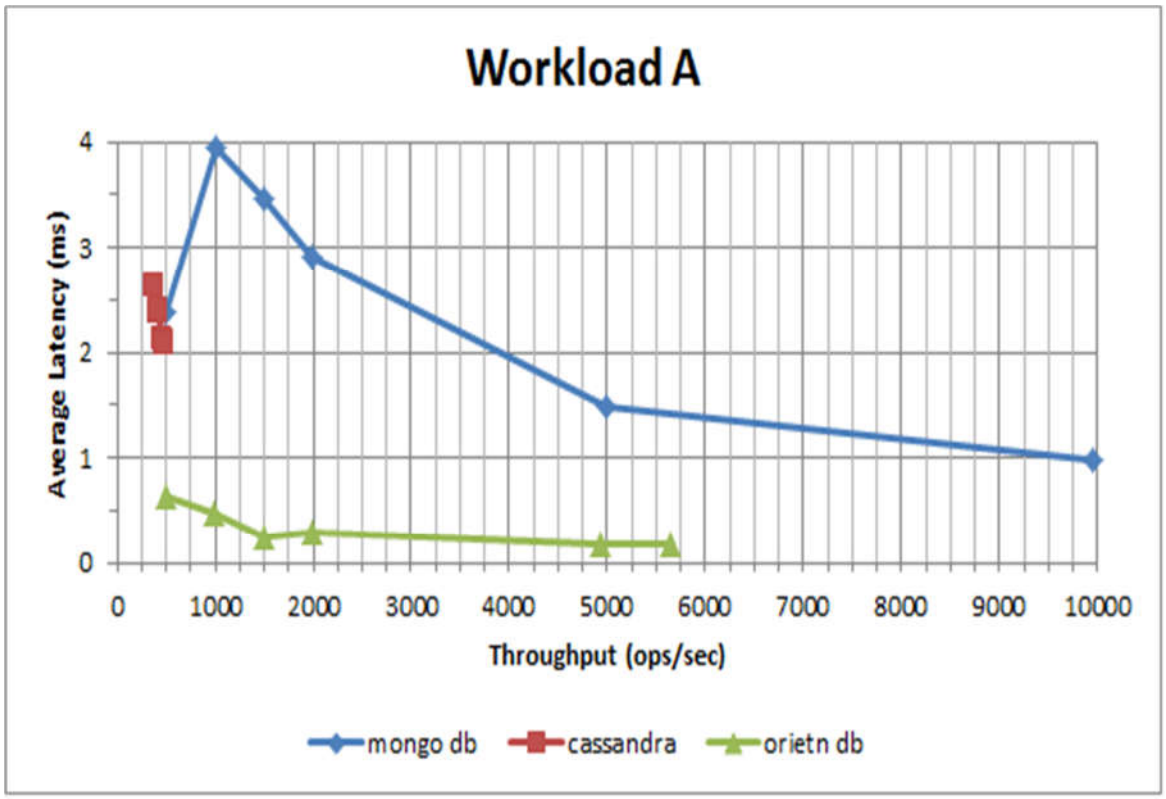
Veritabanı	Throughput	Average Latency
MongoDB	16913,60	5,88
Cassandra	344,23	2,89
OrientDB	3478,52	0,27

6.1.2. İşlemsel evre

Bu evrede bir önceki yükleme evresinde gerçekleşen veri kümelerinin yükleme işlemi sonrasında iş yüklerinin yürütme aşaması gerçekleştirilir. İşlemsel (Throughput) evrede sıralı algoritmamızın kendi içerisinde tutarlı performanslarını yakalayabilmek için YCSB'nin oluşturduğu sıralama algoritmasıyla her bir sistem gerekli testlere tabi tutulmaktadır. Ayrıca yine bu evrede sistemlerin performanslarını daha iyi gözlemleyebilmek adına, sistemlerin saniye de gerçekleştirdiği işlem sayılarına hedefler konularak testler yapılmıştır.

Workload A: Ağır iş yükü güncelleme

Workload A, yüzde 50 oranında okuma ve yüzde 50 oranında yazma karışımı içeren yoğun güncelleme işlemlerinin gerçekleştiği bir senaryodur. Bir kullanıcı oturum açtığında (örneğin bir web uygulamasında), kullanıcı oturumu kapatana kadar veya oturum zaman aşımına uğrayana kadar geçen sürede oluşan oturumla ilgili tüm Oturum verilerini (kullanıcı profili bilgilerini, mesajları, kişiselleştirilmiş verileri ve temaları, önerileri, hedefli promosyonları ve indirimleri) ana bellekte veya oturum deposunda saklar. Workload A iş yükü de, bir kullanıcının son eylemlerini kaydeden bir oturum deposudur. Şekil 6.2'deki Workload A test sonuçların incelediğimiz de, ağır güncelleme senaryoları altında OrientDB'nin ortalama gecikme süresi bakımından en iyi performansı gösterdiği açıkça görülmektedir. MongoDB'nin ise gecikme süresi bakımında her ne kadar Cassandra ve OrientDB'ye kıyasla başarılı bir performans gösteremese de saniye yapılan iş miktarlarında, koyulan 6 hedefi de büyük oranda tutturduğunu gözlemlemekteyiz. Cassandra'nin ise saniye yapılması istenen iş miktarları hedeflerinin tutturmak da kötü bir performans gösterdiğine tanık olduk (Şekil 6.2).Çizelge 6.2'de Workload A grafiğine ait sonuçlar detaylı olarak verilmiştir.



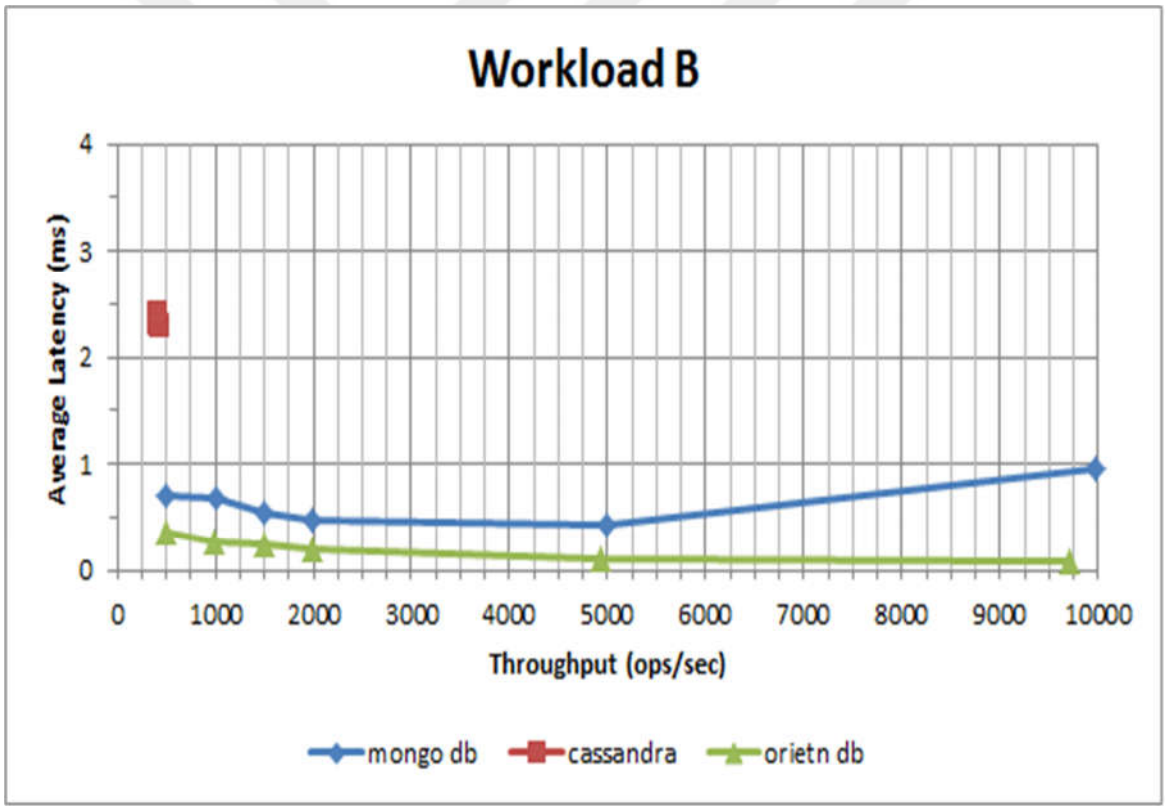
Şekil 6.2. Workload A: Ağır iş yükü güncelleme

Çizelge 6.2 Workload A Throughput ve Average Latency sonuçları

Veritabanı	Target Throughput	Throughput	Average Latency
MongoDB	500	499,85	2,38
	1000	999,54	3,96
	1500	1499,19	3,46
	2000	1998,26	2,90
	5000	4989,84	1,48
	10000	9961,64	0,98
Cassandra	500	376,60	2,64
	1000	417,70	2,38
	1500	467,69	2,12
	2000	471,14	2,10
	5000	473,53	2,10
	10000	411,67	2,41
OrientDB	500	499,28	0,65
	1000	997,00	0,48
	1500	1492,78	0,25
	2000	1988,73	0,30
	5000	4921,79	0,18
	10000	5648,85	0,17

Workload B: İş yükünü çoğunlukla okuma

Workload B iş yükü % 95 oranında okuma ve % 5 oranında güncelleme işlemlerinden oluşur. Fotoğraf etiketleme olayını bu iş yüküne örnek verebiliriz. Etiketleme bir güncelleme işlemidir fakat işlemin çoğu etiketlerin okunmasıdır. Workload B iş yükü sonuçlarını incelediğimiz de, OrientDB ve MongoDB'nin, saniyede gerçekleştirilmesi istenen iş hacimlerini başarılı bir şekilde tutturduklarını ve ayrıca, gecikme sürelerinde de yine birbirine yakın ve başarılı performanslar yakaladıklarını gözlemlemekteyiz (Şekil 6.3). Cassandra ise Workload A dekine benzer vasat bir performans gösterdiğini gözlemlemekteyiz. Çizelge 6.3'de Workload B grafiğine ait sonuçlar detaylı olarak verilmiştir.



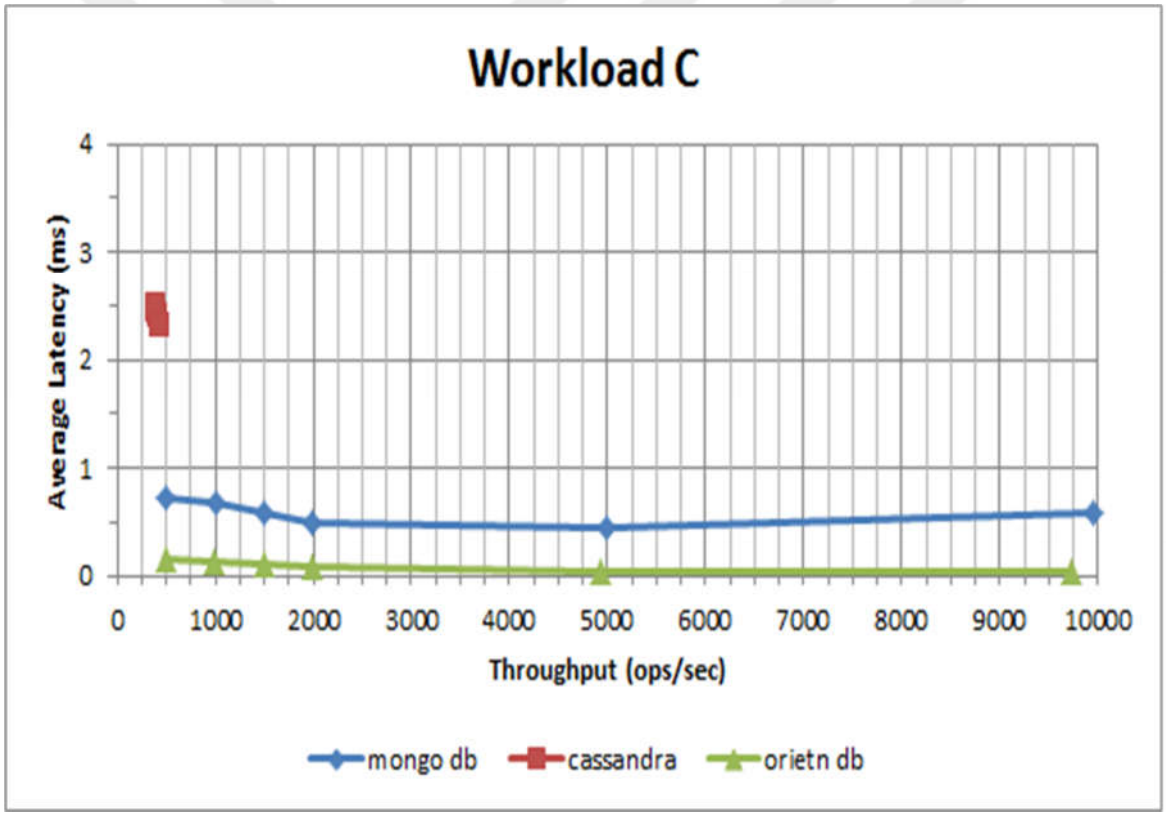
Şekil 6.3. Workload B : İş yükünü çoğunlukla okuma

Çizelge 6.3 Workload B Throughput ve Average Latency sonuçları

Veritabanı	Target Throughput	Throughput	Average Latency
MongoDB	500	499,86	0,699
	1000	999,51	0,694
	1500	1499,03	0,552
	2000	1998,18	0,48
	5000	4991,53	0,439
	10000	9963,93	0,966
Cassandra	500	418,89	2,37
	1000	427,07	2,31
	1500	424,40	2,32
	2000	433,67	2,283
	5000	418,57	2,36
	10000	412,56	2,4
OrientDB	500	499,27	0,35
	1000	997,01	0,279
	1500	1493,27	0,245
	2000	1987,71	0,2
	5000	4922,61	0,104
	10000	9704,30	0,076

Workload C: Sadece okuma

Workload C, %100 oranında okuma işlemine sahiptir. Profillerin başka yerlerde oluşturulduğu kullanıcı profili önbelleği (ör. Hadoop) bu iş yüküne örnek olarak verilebilir. Workload C sonuçları, bize Workload B sonuçlarının birkaç istisnayla tutarlı olduğunu gösterir (Şekil 6.4). Bu iş yükünde de OrientDB ve MongoDB'nin gerçekleştirilmesi istenen iş hacimlerini başarılı bir şekilde tutturduklarını ve Cassandra'nin ise yine ilk 2 iş yükündekine yakın bir performans sergileyerek iş hacminin hedefini arttırırken daha yüksek verim veya düşük gecikme sürelerini elde edemedik. Çizelge 6.4'de Workload C grafiğine ait sonuçlar detaylı olarak verilmiştir.



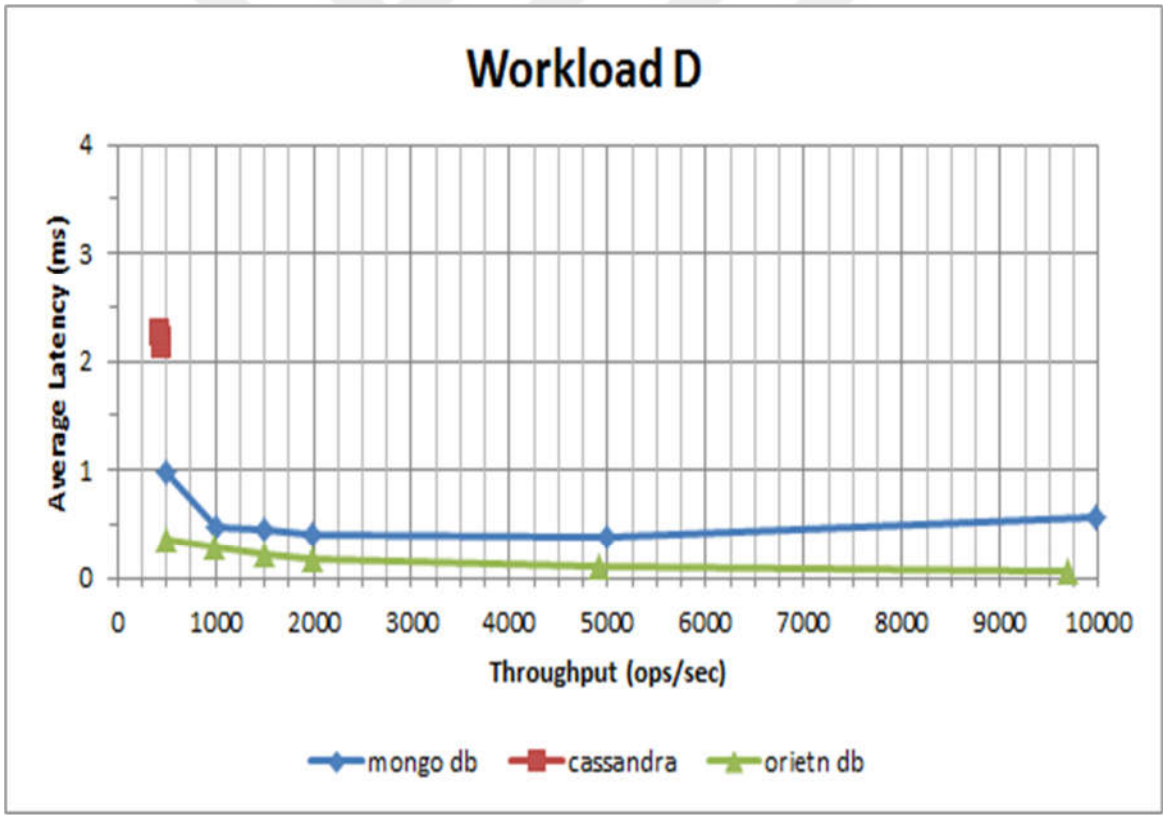
Şekil 6.4. Workload C: Sadece okuma

Çizelge 6.4 Workload C Throughput ve Average Latency Sonuçları

Veritabanı	Target Throughput	Throughput	Average Latency
MongoDB	500	499,85	0,74
	1000	999,52	0,69
	1500	1499,03	0,58
	2000	1998,23	0,49
	5000	4989,42	0,44
	10000	9959,86	0,60
Cassandra	500	429,71	2,31
	1000	406,96	2,44
	1500	428,55	2,32
	2000	415,12	2,40
	5000	399,54	2,49
	10000	431,24	2,31
OrientDB	500	499,22	0,16
	1000	997,20	0,13
	1500	1493,77	0,11
	2000	1988,86	0,09
	5000	4930,65	0,04
	10000	9726,46	0,03

Workload D: Son iş yükünü okuma

Workload D, %95 oranında okuma ve %5 oranında ekleme işleminin gerçekleştirildiği bir iş yüküdür. Günümüzde popülerlikleri git gide artan sosyal medya platformlarındaki, kullanıcıların profillerinde paylaştıkları metin, fotoğraf, video ve animasyonlu GIF vs. gibi paylaşımlarına sürekli veya belirli periyotlarla yenilerini eklemektedirler. Her zaman en son eklenen kayıtlar en popüler olanıdır. Bu iş yükü senaryosu, kullanıcı durumu güncellemelerini veya en son yayını okumak isteyen kullanıcıları simüle eder. OrientDB ve MongoDB'nin bu iş yükündeki Cassandra'ye oranla daha yüksek verimle çalıştığını görmekteyiz (Şekil 6.5). OrientDB gecikme sürelerinden 0,5 ms altında kalarak gayet başarılı bir performansa imza attığını gözlemlemekteyiz. Çizelge 6.5'de Workload D grafiğine ait sonuçlar detaylı olarak verilmiştir.



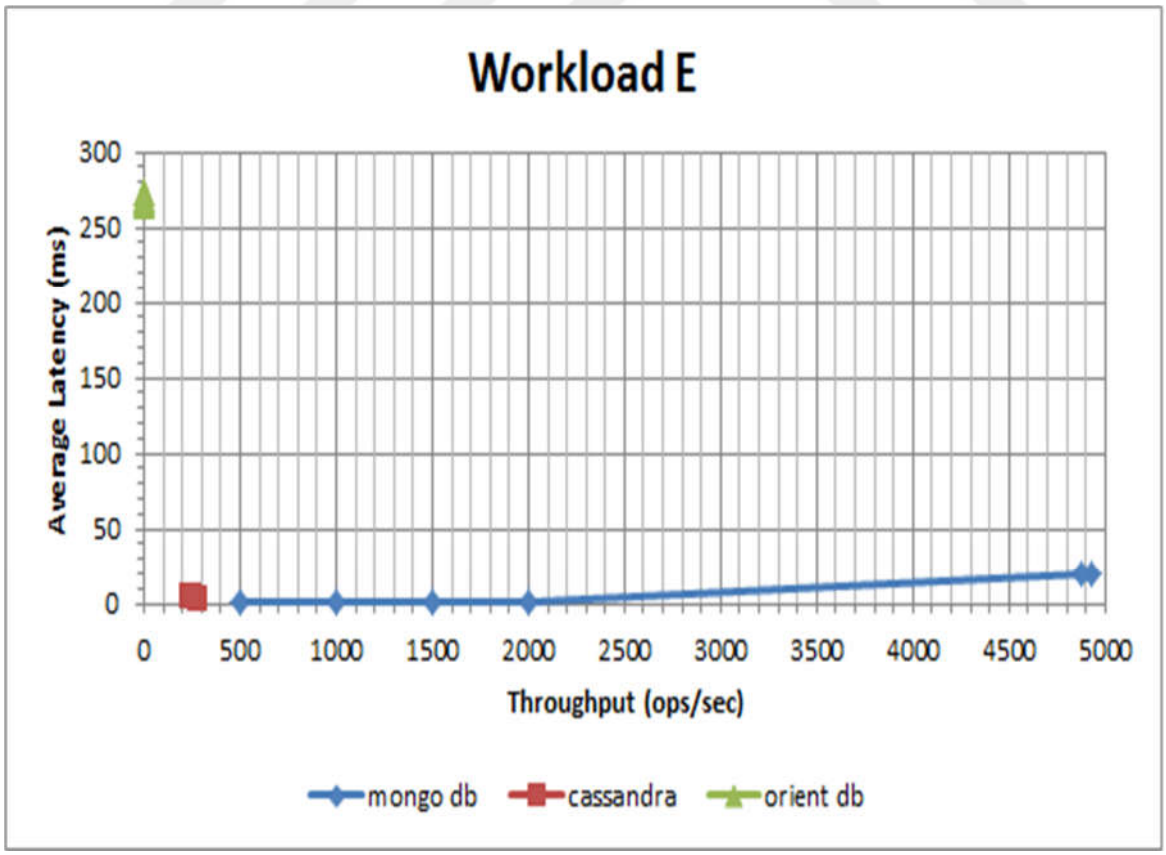
Şekil 6.5. Workload D: Son iş yükünü okuma

Çizelge 6.5 Workload D Throughput ve Average Latency Sonuçları

Veritabanı	Target Throughput	Throughput	Average Latency
MongoDB	500	499,84	0,98
	1000	999,54	0,477
	1500	1499,14	0,446
	2000	1998,21	0,404
	5000	4994,05	0,373
	10000	9969,39	0,56
Cassandra	500	454,72	2,18
	1000	465,94	2,135
	1500	464,31	2,14
	2000	452,87	2,192
	5000	436,52	2,27
	10000	442,81	2,24
OrientDB	500	499,21	0,35
	1000	996,91	0,296
	1500	1493,20	0,23
	2000	1987,69	0,188
	5000	4918,54	0,1
	10000	9688,51	0,063

Workload E: Kısa aralıklar

Workload E iş yükü %95 oranında tarama ve %5 oranında ekleme işlemlerinden oluşur. Bu iş yükünde, bireysel kayıtlar yerine kısa kayıtlar sorgulanır ve işlemlerin % 95'i, kayıt aralıkları üzerinde Arama / Sorgulama gerçekleştiren tarama işlemleridir. Bu iş yüküne, online bir forum da yapılan çevrimiçi bir konuşmadan sonra, iş parçacığı kimliği tarafından kümelenmiş belirli bir iş parçacığındaki iletileri alan tarama işlemlerinin yapıldığı senaryoları örnek olarak verebiliriz. Şekil 6.6'daki tabloyu incelediğimizde, üç veri tabanında bu iş yükünde verimlerinin düşük kaldığını gözlemliyoruz. MongoDB'nin saniyede gerçekleştirilmesi istenen 2000 hedefine kadar verimli çalışabildiği fakat 5000 ve 10000 hedeflerinde çok yüksek gecikme sürelerine çıktığını görmekteyiz. Diğer iş yüklerinde başarılı performanslar sergileyen OrientDB'nin ise bu iş yükünde hem gecikme süreleri olarak hem de saniyede yaptığı işlem sayılarında çok kötü bir performans sergilediğini görüyoruz, Cassandra'nin ise diğer iş yüklerinde sergilediği performansların altında bir görüntü çizdiğini grafiğimizde gözlemliyoruz. Çizelge 6.6'da Workload E grafiğine ait sonuçlar detaylı olarak verilmiştir.



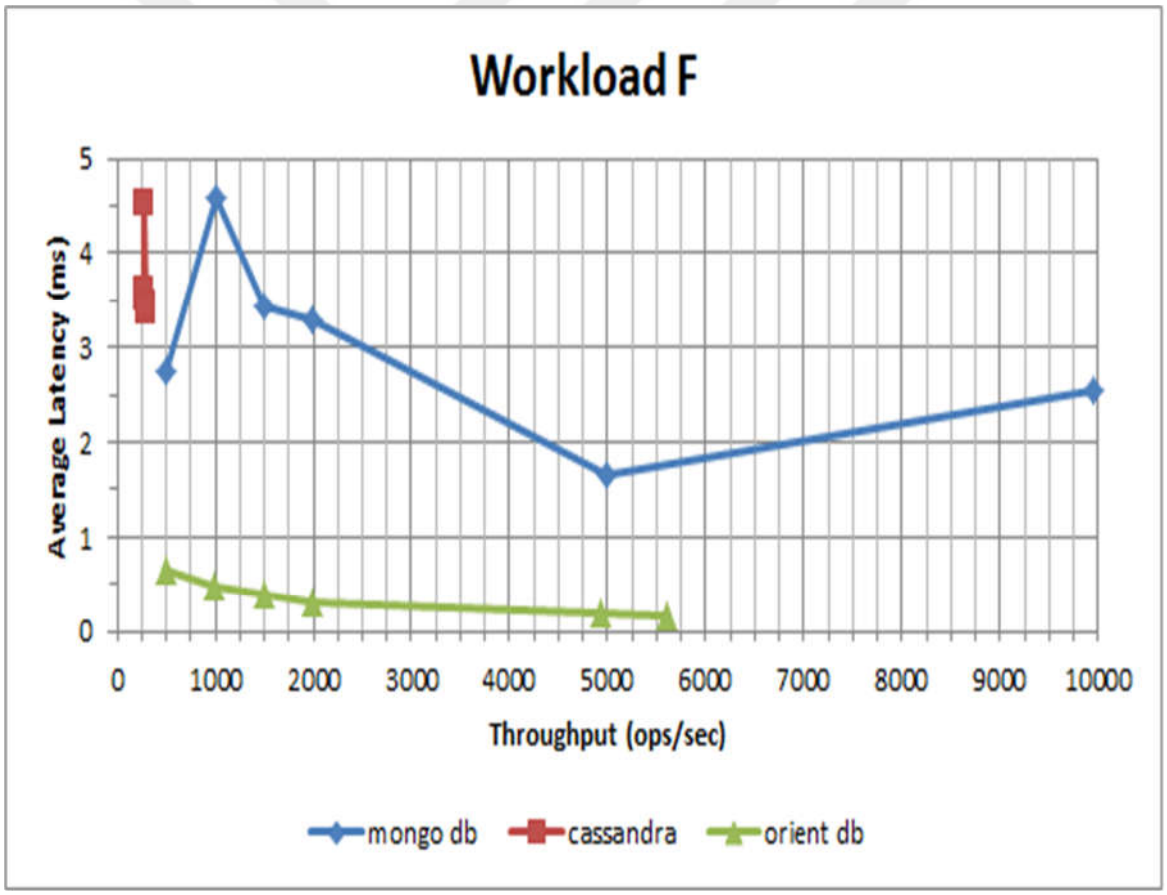
Şekil 6.6. Workload E: Kısa aralıklar

Çizelge 6.6 Workload E Throughput ve Average Latency Sonuçları

Veritabanı	Target Throughput	Throughput	Average Latency
MongoDB	500	499,87	1,294
	1000	999,55	1,170
	1500	1499,04	1,270
	2000	1998,58	1,421
	5000	4925,76	20,270
	10000	4881,54	20,440
Cassandra	500	237,73	4,190
	1000	276,23	3,600
	1500	248,15	4,020
	2000	251,59	3,950
	5000	241,12	4,132
	10000	249,06	4,010
OrientDB	500	3,64	273,630
	1000	3,66	266,980
	1500	3,70	265,090
	2000	3,79	270,780
	5000	3,71	268,880
	10000	3,73	273,650

Workload F: Oku-değiştir-yaz

Workload F iş yükünde, istemcinin bir kaydı okuduğu, değiştirdiği ve değişiklikleri geri yazdığı senaryo ele alınmaktadır. Kullanıcı kayıtlarının kullanıcı tarafından okunup değiştirildiği veya kullanıcının etkinliğini kaydettiği, kullanıcı veri tabanları işlemlerini bu iş yüküne örnek olarak verebiliriz. MongoDB ve OrientDB'nin bu iş yükünde de saniye de yapılması hedeflenen iş hacmini tutturmada en tutarlı performansları gösterdiğini görmekteyiz. Sadece OrientDB'nin 10000 hedefini tutturamadığını fakat gecikme sürelerinde hemen hemen bütün hedeflerde 0.5ms'nin altında kalarak müthiş bir verimlikle çalıştığını söyleyebiliriz (Şekil 6.7). Çizelge 6.7'de Workload F grafiğine ait sonuçlar detaylı olarak verilmiştir.



Şekil 6.7. Workload F: Oku-değiştir-yaz

Çizelge 6.7 Workload F Throughput ve Average Latency Sonuçları

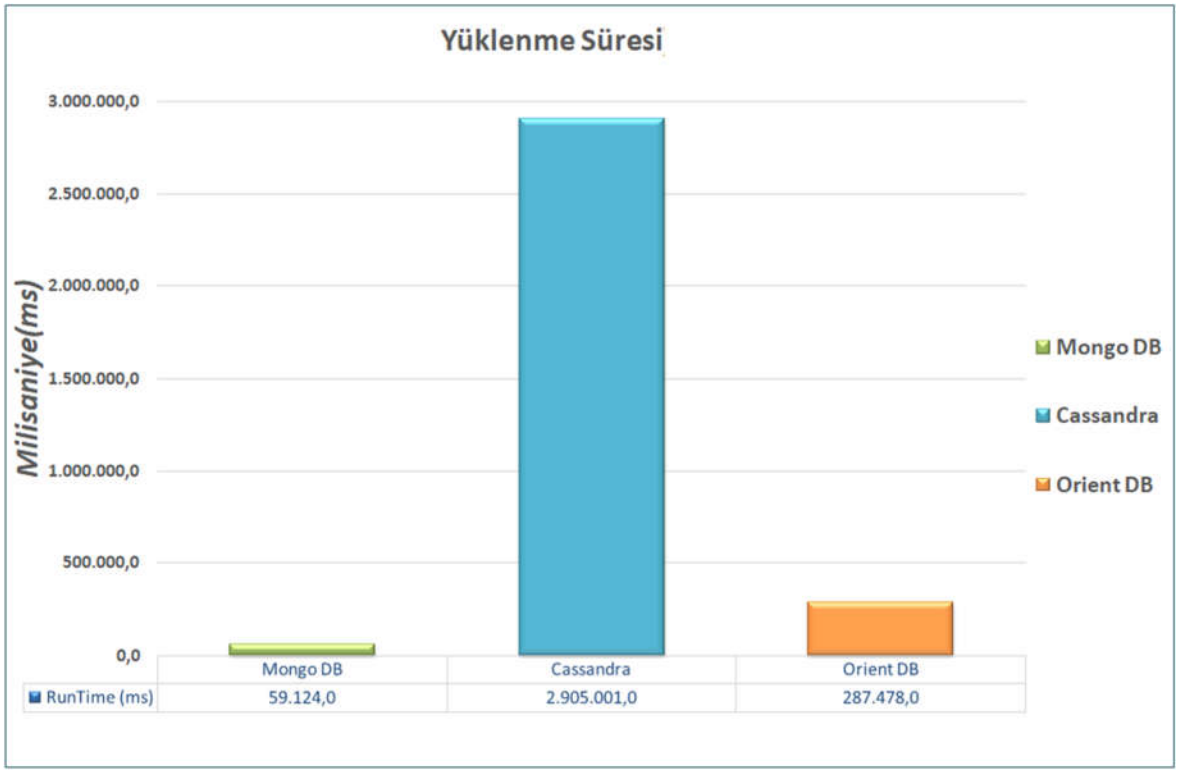
Veritabanı	Target Throughput	Throughput	Average Latency
MongoDB	500	499,87	2,74
	1000	999,51	4,6
	1500	1498,95	3,43
	2000	1998,25	3,295
	5000	4990,79	1,66
	10000	9959,56	2,56
Cassandra	500	283,45	3,51
	1000	287,23	3,46
	1500	279,70	4,53
	2000	274,61	3,62
	5000	292,69	3,37
	10000	289,65	3,43
OrientDB	500	499,28	0,65
	1000	996,93	0,487
	1500	1492,46	0,393
	2000	1986,83	0,32
	5000	4929,56	0,2
	10000	5593,81	0,173

6.2. Runtime Sonuçları

Bu bölümde üç farklı NoSQL sistemine uyguladığımız iş yükleri testlerinden elde edilen çalışma süreleri (Runtime) sonuçları grafikler halinde paylaşılmıştır. Grafikteki Runtime sonuçlarını değerlendirirken, Runtime değerleri düşük olan sistemler başarılı bir sonuç elde etmişlerdir.

6.2.1. Yükleme aşaması

Her bir veritabanına 1 milyon veri yüklenmiştir. Şekil 6.8 bize veritabanların bu verileri sisteme ne kadar sürede yüklendiğini göstermektedir. Grafiği incelediğimizde, MongoDB'nin 1 milyon veriyi sistemine yüklemede, OrientDB'nin ve Cassandra'nın önünde olduğu görülmektedir. Yapılan bu işlemlerin Windows işletim sistemi üzerinde yapıldığı göz ardı edilmemelidir. Yükleme aşamasında (Load Phase) aşamasında iş yüklerini veritabanlarına uygularken, kullanılan 6 iş yüküde benzer veri kümesine sahip olduğu için, her bir iş yükü için sadece "Workload A" parametre dosyaları yüklenmiştir. Yahoo'nun önerdiği algoritmada da bunu gözlemleyebiliyoruz.



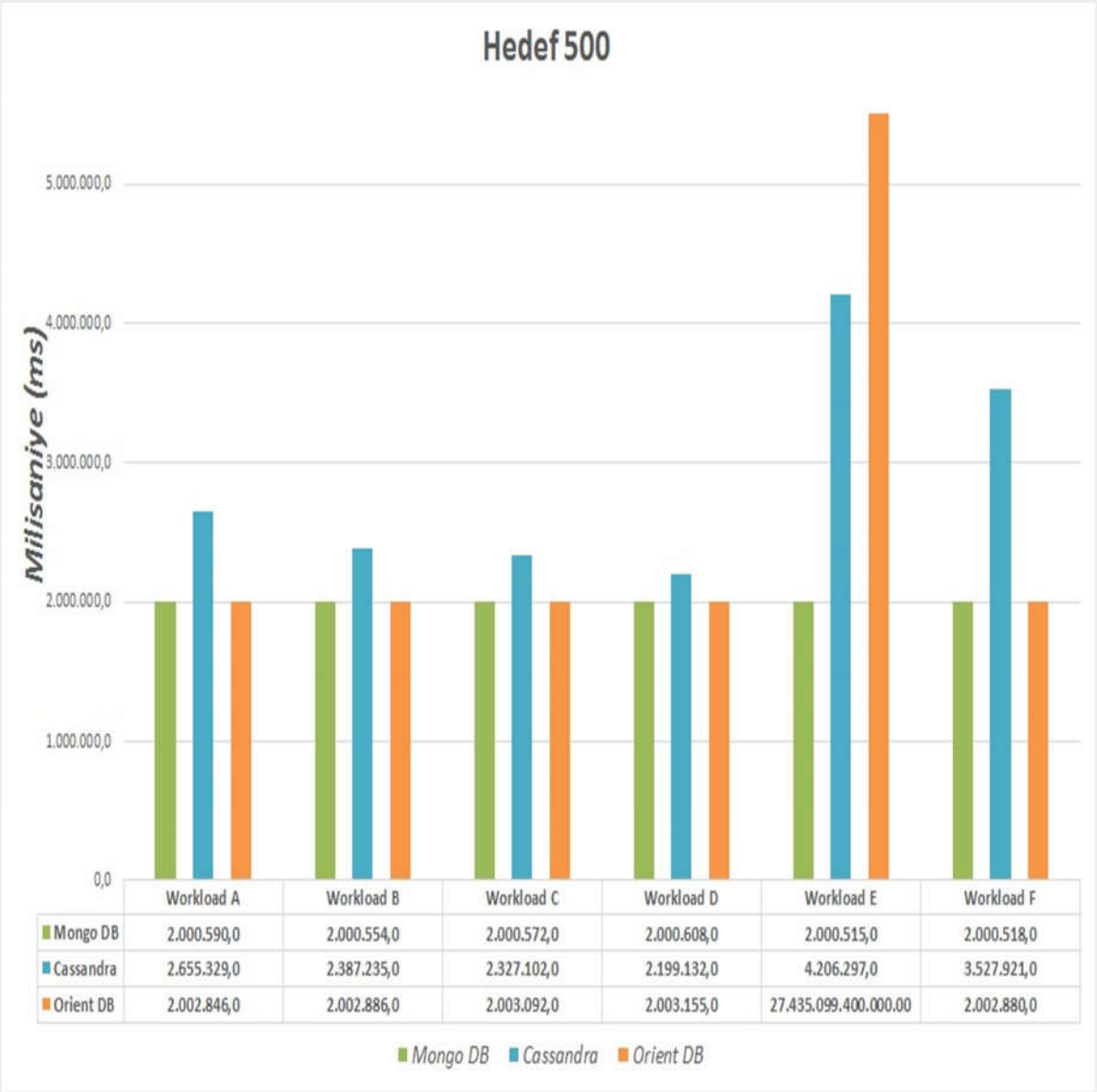
Şekil 6.8. Verilerin yüklenme süreleri

6.2.2. İşlemsel aşama

Bir önceki yükleme aşamasında gerçekleşen veri kümelerinin yükleme işlemi sonrasında iş yüklerinin yürütme aşaması gerçekleştirilir. Bu aşamada sistemlerin performanslarını daha iyi gözlemleyebilmek için sistemlerin saniye de gerçekleştirdiği işlem sayılarına hedefler konularak testler yapılmıştır. MongoDB, Cassandra ve OrientDBNoSQL sistemlerine yükleme aşamasında yüklenen 1000000 kayıt için, bu veri boyutu miktarında her bir veritabanından 1000000 operasyonu gerçekleştirmesi istenmiştir. Gerçekleştirilmesi istenilen 1000000 operasyonu yaparken de, sistemlerin gösterdiği performansları daha iyi test edebilmek amacıyla saniye başına gerçekleştirilmesi istenen hedef iş hacim miktarları konulmuştur. Bu hedef iş hacim miktarları 500, 2 000, 5 000 ve 10 000 olarak ayarlanmıştır.

Hedef 500 işlemi

Hedef 500 aşamasında, her saniyede yapılması hedeflenen bu 500 işlem miktarının kaçını yapabildiğini gözlemlenmiştir. Şekil 6.9'da her bir iş yükünün (Workload A, Workload B, Workload C, Workload D, Workload E, Workload F) ayrı ayrı uygulandığı NoSQL sistemlerin, her bir iş yükü için 500 hedefini gerçekleştirirken, bu işlemi ne kadar sürede gerçekleştirildiğinin süreleri milisaniye cinsinden verilmiştir. Grafiği incelediğimizde Workload E aşaması hariç hedef hacim 500 için, üç sisteminde birbirlerine yakın performans sergilediklerini söyleyebiliriz. Workload E iş yükü için ise MongoDB'nin 1 milyon operasyonunu belirlenen iş hacminde gerçekleştirmede, yaklaşık olarak Cassandra'dan 2 kat daha kısa sürede işlemi gerçekleştirdiği gözlemlenmiştir. Orient DB ise Workload E iş yükünün istenilen operasyonu 3 gün gibi bir sürede yaparak bu iş yükü için kötü bir performans sergilemiştir.

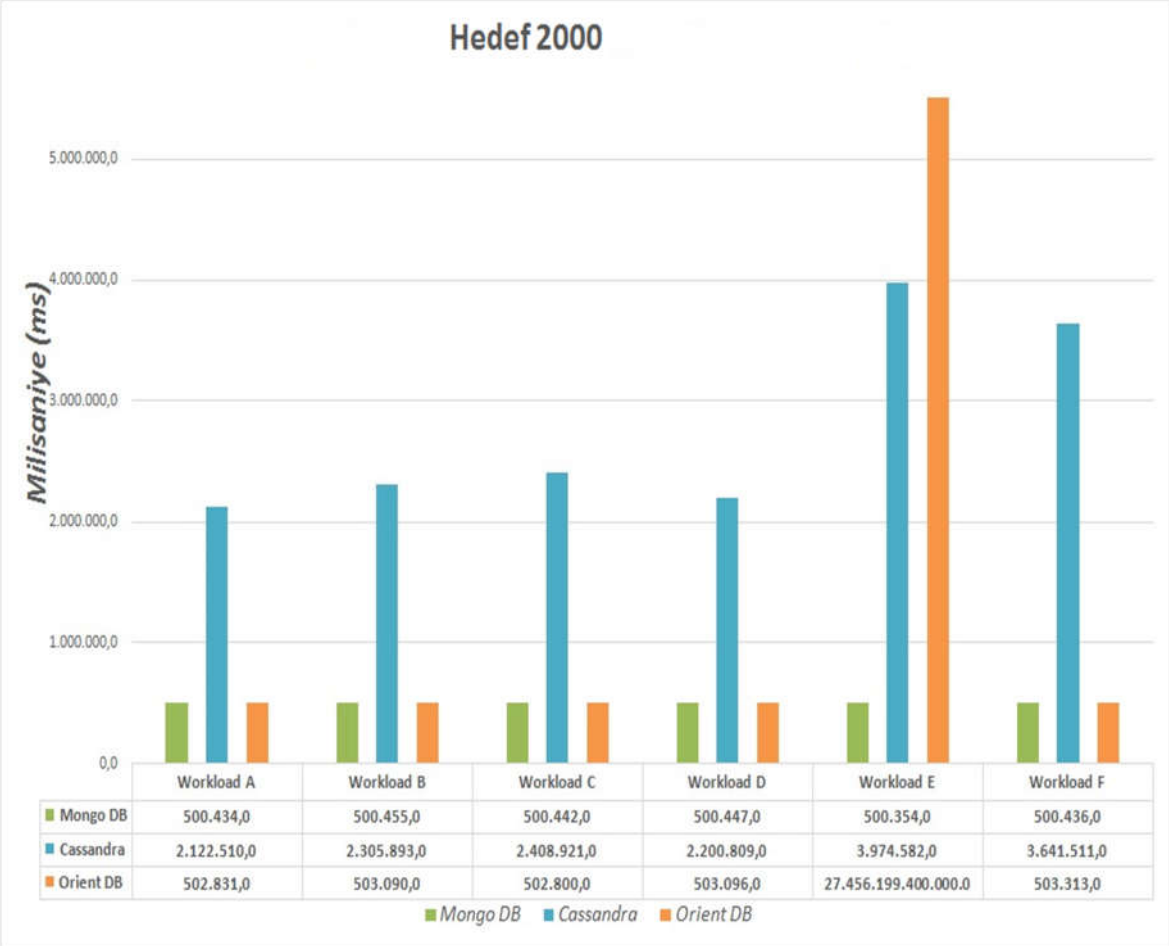


	Workload A	Workload B	Workload C	Workload D	Workload E	Workload F
Mongo DB	2.000.590,0	2.000.554,0	2.000.572,0	2.000.608,0	2.000.515,0	2.000.518,0
Cassandra	2.655.329,0	2.387.235,0	2.327.102,0	2.199.132,0	4.206.297,0	3.527.921,0
Orient DB	2.002.846,0	2.002.886,0	2.003.092,0	2.003.155,0	27.435.099.400.000.00	2.002.880,0

Şekil 6.9. Hedef 500 için yükleme yürütme süreleri

Hedef 2 000 işlemi

Hedef 2000 aşamasında, her saniyede yapılması hedeflenen bu 2000 işlem miktarının kaçını yapabildiğini gözlemlenmiştir. Şekil 6.10'daki grafikte her bir iş yükünün ayrı ayrı uygulandığı NoSQL sistemlerin, her bir iş yükü için 2000 hedefini gerçekleştirirken, bu işlemi ne kadar sürede gerçekleştirildiğinin süreleri milisaniye cinsinden verilmiştir.



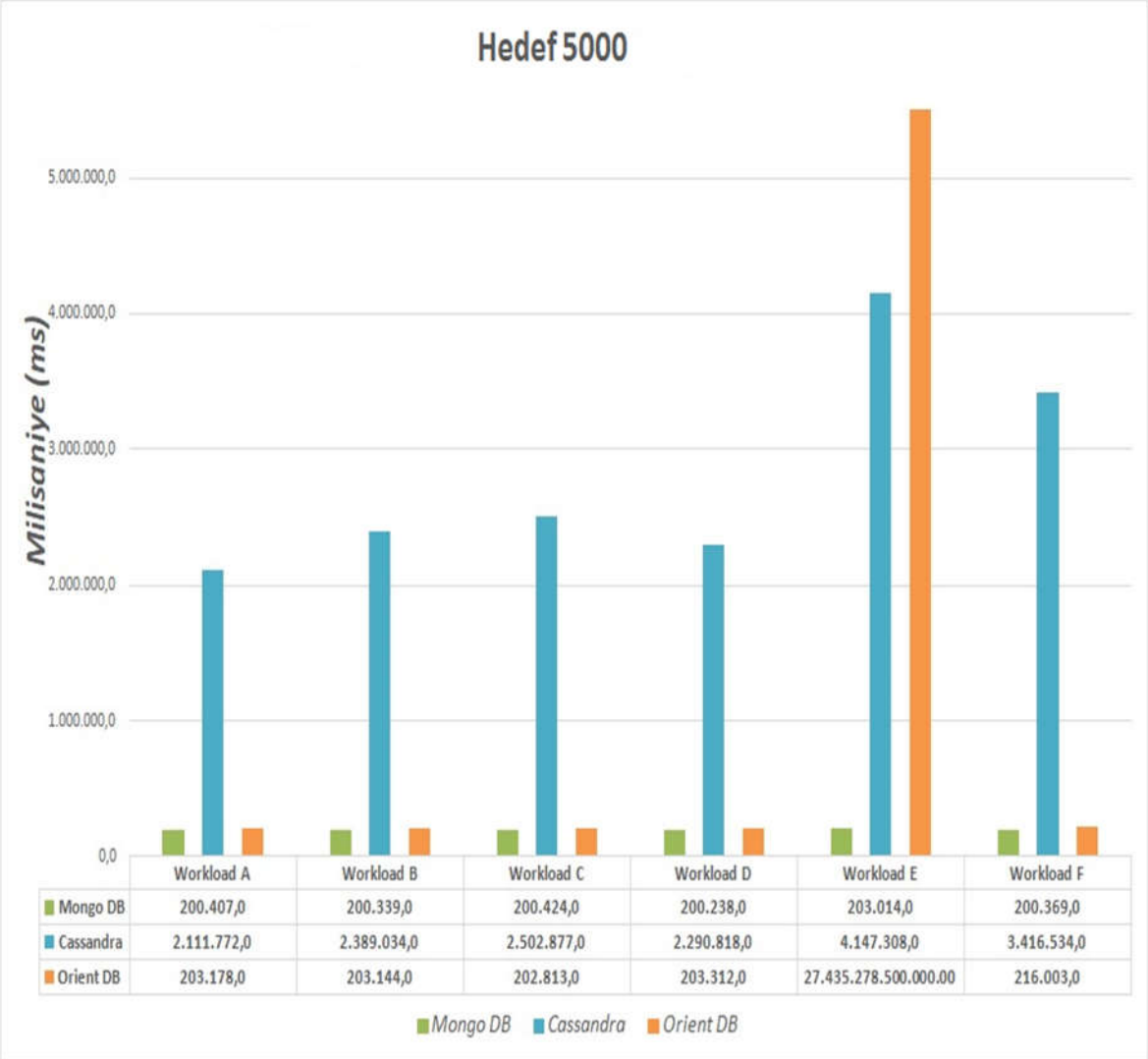
	Workload A	Workload B	Workload C	Workload D	Workload E	Workload F
Mongo DB	500.434,0	500.455,0	500.442,0	500.447,0	500.354,0	500.436,0
Cassandra	2.122.510,0	2.305.893,0	2.408.921,0	2.200.809,0	3.974.582,0	3.641.511,0
Orient DB	502.831,0	503.090,0	502.800,0	503.096,0	27.456.199.400.000,0	503.313,0

Şekil 6.10. Hedef 2 000 için yükleme yürütme süreleri

Grafiği incelediğimizde MongoDB'nin ve OrientDB'nin 1 milyon operasyonu, belirlenen iş hacminde gerçekleştirirken geçen süre de, Workload E iş yükü hariç, birbirlerine yakın performanslar sergilediğini gözlemleyebiliyoruz. Cassandra ise Hedef 500'deki performansına benzer performans sürelerinde kaldığını gözlemliyoruz. Bunun sebebi olarak ise saniye de 2000 hedefi konulmasına rağmen sadece yaklaşık olarak 500'ünü yapabildiği için Cassandra'nin, Hedef 500 grafiğindeki benzer bir performans gösterdiğini gözlemliyoruz.

Hedef 5 000 işlemi

Hedef 5000 aşamasında sistemlerin, her saniyede yapılması hedeflenen bu 5000 işlem miktarının kaçını yapabildiği gözlemlenmiştir. Şekil 6.11'de her bir iş yükünün ayrı ayrı uygulandığı NoSQL sistemlerin, her bir iş yükü için 5000 hedefini gerçekleştirirken, bu işlemi ne kadar sürede gerçekleştirildiğinin süreleri milisaniye cinsinden verilmiştir. Grafiği incelediğimizde MongoDB'nin ve OrientDB'nin 1 milyon operasyonu, belirlenen iş hacminde gerçekleştirmede, Workload E iş yükü hariç, birbirlerine yakın performanslar sergilediğini gözlemleyebiliyoruz. MongoDB ve OrientDB ise hedeflenen iş hacimlerini bu aşamada da tutturabildikleri için Cassandra'ye süre olarak büyük farklar atmışlardır.

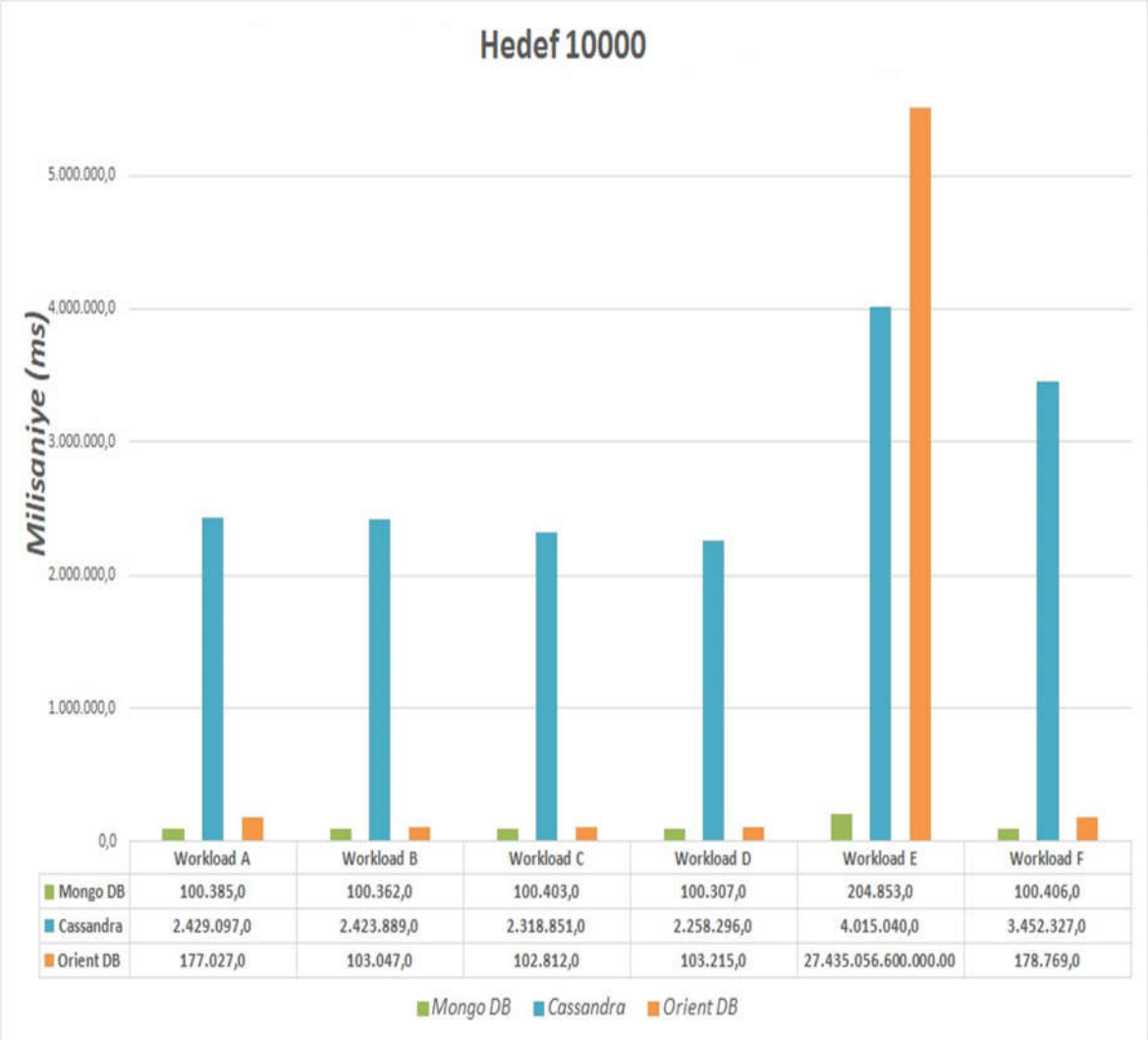


	Workload A	Workload B	Workload C	Workload D	Workload E	Workload F
Mongo DB	200.407,0	200.339,0	200.424,0	200.238,0	203.014,0	200.369,0
Cassandra	2.111.772,0	2.389.034,0	2.502.877,0	2.290.818,0	4.147.308,0	3.416.534,0
Orient DB	203.178,0	203.144,0	202.813,0	203.312,0	27.435.278.500.000.00	216.003,0

Şekil 6.11. Hedef 5 000 için yükleme yürütme süreleri

Hedef 10 000 işlemi

Hedef 10000 aşamasında sistemlerin, her saniyede yapılması hedeflenen bu 10000 işlem miktarının kaçını yapabildiği gözlemlenmiştir. Şekil 6.12'de her bir iş yükünün ayrı ayrı uygulandığı NoSQL sistemlerin, her bir iş yükü için 10000 hedefini gerçekleştirirken, bu işlemi ne kadar sürede gerçekleştirdiğini vermektedir.



	Workload A	Workload B	Workload C	Workload D	Workload E	Workload F
Mongo DB	100.385,0	100.362,0	100.403,0	100.307,0	204.853,0	100.406,0
Cassandra	2.429.097,0	2.423.889,0	2.318.851,0	2.258.296,0	4.015.040,0	3.452.327,0
Orient DB	177.027,0	103.047,0	102.812,0	103.215,0	27.435.056.600.000.00	178.769,0

Şekil 6.12. Hedef 10 000 için yükleme yürütme süreleri

Grafiği incelediğimizde MongoDB'nin ve OrientDB'nin, Workload E iş yükü hariç, kabul edilebilir seviyede performanslar sergilediğini gözlemliyoruz. Cassandra'nin performansı ise diğer grafikler de de karşılaştığımız duruma benzer bir görünümde ve MongoDB ve OrientDB'nin gerisinde kaldığını gözlemliyoruz.

7. SONUÇLAR

Teknoloji dünyasının her alanında yaşanan hızlı gelişimin sonucu olarak ortaya çıkan büyük veri kavramı gün geçtikçe önüne geçilemez bir şekilde artmaya devam etmektedir. Yaşanan bu değişimle beraber büyük veri binlerce kaynaktan beslenerek, yapılandırılmış, yarı yapılandırılmış ve yapılandırılmamış gibi farklı şekillerde genişlemeye devam etmektedir. Bu devasa boyuttaki verileri bir problem olarak görmek yerine bu verileri kullanarak yapılan çıkarımlar sayesinde hemen hemen her sektör için büyük fırsatlar yakalayabiliriz. Bugün bu veriler kullanılarak, insanlığın en temel ihtiyacı olan sağlık sektörü için birçok çalışma yapabiliriz. Büyük veriler işlenerek insanların yaşlarına göre hastalık eğilimleri ne yönde ilerliyor konusunda çıkarımlar yapıp, bu sayede insanların hastalık oluşmadan önce önlem alması sağlanır. Sadece sağlık alanında değil, her sektörün büyük veriyi göz ardı etmeden bir sorun olarak görmek yerine nasıl fayda sağlayacağını düşüneceğiyle ilgili olarak yeni çözüm arayışlarına girmesi her sektörün kendi yararına bir durumdur. Bu çalışmada büyük verinin 5V olarak geçen bileşenlerinden hacim (Volume), hız (Velocity) ve çeşitlilik (Variety), gerçeklik (Veracity), değer (Value) açıklanarak, bu bileşenlerin doğru olarak anlaşılması neticesinde büyük verinin insanların faydasına kullanılması hususunda sektörlerin doğru adımlar atması amaçlanmıştır.

Günümüzde verilerin ulaştıkları boyutlar itibariyle büyük veri olarak adlandırılan yapılar oluşmuş ve bu devasa miktardaki veri yığınlarını, sektörün tek hâkimi olarak ilerleyen ilişkisel veri tabanları sistemleriyle işlemek ve yönetmek imkânsız hale gelmiştir. Bu noktada birçok firma farklı sektörlerde kullanılmak üzere yüzlerce yeni veri tabanı yönetim sistemleri üretmişlerdir. Nosql olarak adlandırılan bu sistemleri iyi, kötü veya yeterli, yetersiz sistemler olarak kıyaslamaktan çok her sektörün kendi ihtiyacına uygun sistemi bulması adına daha yapıcı kıyaslamalar yapılması NoSQL sistemlerin gelişmesi adına daha fazla yarar sağlar. Büyük Veri dünyasında, her bir kullanıcının gereksinimlerini karşılayabilecek farklı özelliklere ve yeteneklere sahip birçok veri tabanı sistemi mevcuttur.

Bu çalışmada, üç farklı NoSQL veri tabanı sistemini, yeteneklerini ve farklı operasyonlarda nasıl tepki verdiklerini ortaya koymak için tartıştık ve test ettik. Çalışmamızda, Yahoo'nun veri tabanı performanslarını test etmek için tasarladığı bir çerçeve olan Yahoo Cloud Serving Benchmark'ı (YCSB) kullandık. Test ettiğimiz her bir

veri tabanının mimarisi ve tasarımı nedeniyle, her bir işlem için veri tabanların farklı performans sonuçlarını verdiğini gözlemledik. Elde edilen sonuçlara göre, MongoDB'nin saniye de gerçekleştirilmesi istenen hedefleri yakalamada en başarılı performans sonuçlarına ulaştığı gözlemledik, fakat bunun yanında OrientDB'ye oranla gecikme sürelerin de daha kötü bir performans gerçekleştirdiğini gördük. Cassandra ise hemen hemen bütün testlerde, saniye de gerçekleştirilmesi istenen hedefleri yakalamada ve her bir işlem arasındaki gecikme süresi performanslarında MongoDB ve OrientDB'nin gerisinde kalmıştır.

Sonuç olarak bu çalışma, her biri kendi alanın öncü konumunda olan üç farklı NoSQL veri tabanı sisteminin arasında performans karşılaştırmasının yapılması ve iş yüklerine dayalı olarak oluşan farklı durum ve koşulların altında veri tabanlarının nasıl etkilediklerini ve nasıl sonuçlar verdiklerini açığa çıkarmaktır. Bu sayede yazılımcılar çalışmadaki sonuçları inceleyerek, piyasada bulunan yüzlerce veri tabanı teknolojileri arasından neden NoSQL teknolojisini seçmesi konusunda ve ayrıca kendi çalışmaları için en uygun NoSQL veri tabanı sisteminin ne olduğuna karar verme hususunda çıkarımlarda bulunabilirler.

KAYNAKLAR

[1] S. ÖZTÜRK and H. E. ATMACA, "İlişkisel ve İlişkisel Olmayan (NoSQL) Veri Tabanı Sistemleri Mimari Performansının Yönetim Bilişim Sistemleri Kapsamında İncelenmesi," *Bilişim Teknol. Derg.*, pp. 199–199, 2017.

[2] V. C. Storey and I. Y. Song, "Big data technologies and Management: What conceptual modeling can do," *Data Knowl. Eng.*, vol. 108, no. February, pp. 50–67, 2017.

[3] M. A. Khan, M. F. Uddin, and N. Gupta, "Seven V'S of Big Data," 2014.

[4] M. Ge, H. Bangui, and B. Buhnova, "Big Data for Internet of Things: A Survey," *Futur. Gener. Comput. Syst.*, vol. 87, pp. 601–614, 2018.

[5] S. Khan, X. Liu, K. A. Shakil, and M. Alam, "A survey on scholarly data: From big data perspective," *Inf. Process. Manag.*, vol. 53, no. 4, pp. 923–944, 2017.

[6] A. Ribeiro, A. Silva, and A. R. da Silva, "Data Modeling and Data Analytics: A Survey from a Big Data Perspective," *J. Softw. Eng. Appl.*, vol. 08, no. 12, pp. 617–634, 2015.

[7] D. Ganesh Chandra, "BASE analysis of NoSQL database," *Futur. Gener. Comput. Syst.*, vol. 52, pp. 13–21, 2015.

[8] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler, "The Hadoop distributed file system," *Proc. IEEE 26th Symp. Mass Storage Syst. Technol.*, vol. 1, pp. 1–10, 2010.

[9] T. White, *Hadoop : The Definitive Guide*. 2012.

[10] Sharma, Tim, Wong, Gadia, and Sharma, "A brief review on leading big data models," *Data Sci. J.*, vol. 13, no. December, pp. 138–157, 2014.

[11] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking Cloud Serving Systems with YCSB," *Proc. 1st ACM Symp. Cloud Comput.*, 2010.

[12] A. Colaso *et al.*, "Memory Hierarchy Characterization of NoSQL Applications through Full-System Simulation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 5, pp. 1161–1173, 2018.

[13] C. J. Choi, "A Study and Comparison of NoSql Databases," no. May, 2014.

[14] B. Cabral, J. R. Lourenço, M. Vieira, and J. Bernardino, "Comparing NoSQL Databases with a Relational Database: Performance and Space," *Serv. Trans. Big Data*, vol. 2, no. 1, pp. 1–14, 2018.

[15] M. M. Shwaysh, "Security and Performance Comparison of NoSQL Database Systems," no. July, p. 122, 2018.

[16] S. Sagiroglu and D. Sinanc, "Big data: A review," *Collab. Technol. Syst. (CTS), 2013 Int. Conf.*, pp. 42–47, 2013.

[17] G. Wang and J. Tang, "The NoSQL principles and basic application of cassandra model," *Proc. - 2012 Int. Conf. Comput. Sci. Serv. Syst. CSSS 2012*, pp. 1332–1335, 2012.

[18] S. Gilbert and N. Lynch, "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services," *ACM SIGACT News*, vol. 33, no. 2, pp. 51–59, 2005.

[19] A B M Moniruzzaman and S. A. Hossain, "NoSQL Database: New Era of Databases for Big data Analytics," *Int. J. Database Theory Appl.*, vol. 6, no. 4, pp. 1–14, 2013.

[20] D. Pritchett, "Base: an Acid Alternative," *Queue*, vol. 6, no. 3, pp. 48–55, 2008.

[21] W. Vogels, "Eventually Consistent," *Commun. acm*, vol. 52, no. 1, pp. 40–44, 2009.

[22] Deepak GC, “A Critical Comparison of NOSQL Databases in the Context of Acid and Base,” *Culminating Proj. Inf. Assur.*, p. 8, 2016.

[23] Giuseppe DeCandia *et al.*, “Dynamo: Amazon’s Highly Available Key-value Store,” *SOSP 07 Proc. twenty-first ACM SIGOPS*, pp. 205–207, 2007.

[24] A. S. Perumal Murugan, “A Study of NoSQL and NewSQL databases for data aggregation on Big Data,” 2013.

[25] MongoDB, “MongoDB web page,” *Mongo DB Inc.*, 2009. [Online]. Available: http://www.mongodb.com/. [Accessed: 15-Jan-2018].

[26] MongoDB, “Mongo DB Documentation 3.6,” *Mongo DB Inc.*, 2017. [Online]. Available: https://docs.mongodb.com/v3.6. [Accessed: 05-Mar-2018].

[27] K. Bhamra, “A Comparative Analysis of MongoDB and Cassandra,” The University of Bergen, 2017.

[28] L. Kumar, D. S. Rajawat, and K. Joshi, “Comparative analysis of NoSQL (MongoDB) with MySQL Database,” *Int. J. Mod. Trends Eng. Res.*, vol. 2, no. 5, pp. 120–127, 2015.

[29] Cassandra, “Cassandra web page,” *Apache Software Foundation*, 2008. [Online]. Available: http://cassandra.apache.org/. [Accessed: 18-Jan-2018].

[30] B. ERDEN, “NOSQL VERİTABANLARI İÇİN ORTAK SORGU İŞLETİM KATMANININ GELİŞTİRİMİ,” EGE ÜNİVERSİTESİ, 2018.

[31] Cassandra, “Apache Cassandra Documentation v3.4,” *Apache Software Foundation*. [Online]. Available: http://cassandra.apache.org/doc. [Accessed: 01-Jun-2018].

[32] E. Hewitt and J. Carpenter, *Cassandra: The Definitive Guide*. O’Reilly Media, 2016.

[33] A. Lakshman and P. Malik, “Cassandra - A Decentralized Structured Storage System,” *Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, 2010.

[34] OrientDB, “OrientDB web page,” *OrientDB Ltd*, 2010. [Online]. Available: https://orientdb.com/. [Accessed: 03-Sep-2018].

[35] OrientDB, “OrientDB Manual - version 2.0,” *OrientDB Ltd*, 2017. [Online]. Available: http://www.orientechnologies.com/docs/2.0/OrientDB-Manual.pdf. [Accessed: 07-Sep-2018].

[36] D. Fernandes and J. Bernardino, “Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB,” *Proc. 7th Int. Conf. Data Sci. Technol. Appl.*, pp. 373–380, 2018.

[37] Hazelcast, “Hazelcast web page,” *Hazelcast, Inc.*, 2008. [Online]. Available: https://hazelcast.com/. [Accessed: 01-Oct-2018].

[38] A. H. Hammood, “A comparision of NoSQL database systems: A study on MongoDB, Apache, HBase, and Apache Cassandra,” ÇANKAYA ÜNİVERSİTESİ, 2016.

[39] S. R. ALI, “A Comparison of SQL and NoSQL Databases,” ATILIM UNIVERSITY, 2018.

[40] A. T. Aladily, “THE PERFORMANCE –WISE COMPARISON OF THE MOST WIDELY USED NOSQL DATABASES,” KADIR HAS UNIVESITY, 2015.

[41] B. Cooper, “Ycsb core workloads,” *GitHub*, 2010. [Online]. Available: https://github.com/brianfrankcooper/YCSB/wiki/Core-Workloads. [Accessed: 01-Oct-2017].

ÖZGEÇMİŞ

Kişisel Bilgiler

Soyadı, adı :Kara, Burak Cem

Uyruğu :T.C.

Doğum tarihi ve yeri :18.04.1991, Hatay

Medeni hali :Bekar

Telefon :-

Faks :-

e-mail :ckara7@hho.edu.tr

Eğitim

Derece	**Eğitim Birimi**	**Mezuniyet Tarihi**
Yüksek lisans	İskenderun Teknik Üniversitesi/ Elektrik-Elektronik Mühendisliği	Devam ediyor
Lisans	Namık Kemal Üniversitesi/Bilgisayar Mühendisliği	2013
Lise	Büyükşehir Hüseyin Yıldız Anadolu Lisesi	2009

İş Deneyimi

Yıl	Yer	Görev
2019-Halen	HAVA HARP AKADEMİSİ	Subay/Mühendis
2015-2019	KAPAKLI BELEDİYESİ	Mühendis

Yabancı Dil

İngilizce

Yayınlar

B.C. Kara, Y. Daşdemir, Büyük Veri Çözümlerinde Bulut Tabanlı Performans Analizi, 4th International Mediterranean Science and Engineering Congress (IMSEC 2019) .

L.Gökrem, S. Güzel, B.C. Kara, Y. Daşdemir, M. Kara, Bulut Bilişim Üzerinden Akıllı Taşıma Sistemi, 4th International Mediterranean Science and Engineering Congress (IMSEC 2019) .

DİZİN

TEKNOVERSİTE

İSKENDERUN TEKNİK ÜNİVERSİTESİ

teknoversite AYRICALIĞINDASINIZ

İSTE