

T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

NOSQL VERİTABANI SİSTEMLERİNİN PERFORMANS
KARŞILAŞTIRILMASI VE ANALİZİ

YÜKSEK LİSANS TEZİ

Süleyman ÖNDER

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Bilim Dalı

EKİM 2024

T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

NOSQL VERİTABANI SİSTEMLERİNİN PERFORMANS
KARŞILAŞTIRILMASI VE ANALİZİ

YÜKSEK LİSANS TEZİ

Süleyman ÖNDER

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Bilim Dalı

Tez Danışmanı: Dr. Öğr. Üyesi Abdullah SEVİN

EKİM 2024

Süleyman ÖNDER tarafından hazırlanan “NOSQL VERİTABANI SİSTEMLERİNİN PERFORMANS KARŞILAŞTIRILMASI VE ANALİZİ” adlı tez çalışması 16.06.2022 tarihinde aşağıdaki jüri tarafından oy birliği ile Sakarya Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim **Dalı** **Bilgisayar Mühendisliği Bilim** Dalı’nda **Yüksek Lisans tezi**olarak kabul edilmiştir.

Tez Jürisi

Jüri Başkanı :	Unvan Adı SOYADI Sakarya Üniversitesi	
Jüri Üyesi :	Unvan Adı SOYADI (Danışman) Sakarya Üniversitesi	
Jüri Üyesi :	Unvan Adı SOYADI Sakarya Üniversitesi	



ETİK İLKE VE KURALLARA UYGUNLUK BEYANNAMESİ

Sakarya Üniversitesi Fen Bilimleri Enstitüsü Lisansüstü Eğitim-Öğretim Yönetmeliğine ve Yükseköğretim Kurumları Bilimsel Araştırma ve Yayın Etiği Yönergesine uygun olarak hazırlamış olduğum “NOSQL VERİTABANI SİSTEMLERİNİN PERFORMANS KARŞILAŞTIRILMASI VE ANALİZİ” başlıklı tezin bana ait, özgün bir çalışma olduğunu; çalışmamın tüm aşamalarında yukarıda belirtilen yönetmelik ve yönergeye uygun davrandığımı, tezin içerdiği yenilik ve sonuçları başka bir yerden almadığımı, tezde kullandığım eserleri usulüne göre kaynak olarak gösterdiğimi, bu tezi başka bir bilim kuruluna akademik amaç ve unvan almak amacıyla vermediğimi ve 20.04.2016 tarihli Resmi Gazete’de yayımlanan Lisansüstü Eğitim ve Öğretim Yönetmeliğinin 9/2 ve 22/2 maddeleri gereğince Sakarya Üniversitesi’nin abonesi olduğu intihal yazılım programı kullanılarak Enstitü tarafından belirlenmiş ölçütlere uygun rapor alındığını, etik kurul onay belgesi aldığımı, çalışmamla ilgili yaptığım bu beyana aykırı bir durumun ortaya çıkması halinde doğabilecek her türlü hukuki sorumluluğu kabul ettiğimi beyan ederim.

(10/10/2024).

(imza)

Süleyman ÖNDER





Aileme ve Bugünlerime Katkısı Olan Herkese,



TEŞEKKÜR

Yüksek lisans eğitimim boyunca değerli bilgi ve deneyimlerinden yararlandığım, her konuda bilgi ve desteğini almaktan çekinmediğim, araştırmanın planlanmasından yazılmasına kadar tüm aşamalarında yardımlarını esirgemeyen, teşvik eden, aynı titizlikte beni yönlendiren değerli danışman hocam Dr. Öğr. Üyesi Abdullah SEVİN'e teşekkürlerimi sunarım.

Eğitimim ve tez yazım sürecim boyunca her türlü desteği sağlayan, yardımlarını esirgemeyen Sakarya Üniversitesi Bilgisayar Mühendisliği bölüm hocalarıma teşekkür ederim.

Ve tüm eğitim hayatım boyunca benden maddi ve manevi desteklerini esirgemeyen her zaman yanımda olan kıymetli aileme teşekkürlerimi bir borç bilirim.

Süleyman ÖNDER



İÇİNDEKİLER

Sayfa

ETİK İLKE VE KURALLARA UYGUNLUK BEYANNAMESİ	v
TEŞEKKÜR	ix
İÇİNDEKİLER	xi
KISALTMALAR	xv
SİMGELER	xvii
TABLO LİSTESİ	xix
ÖZET.....	xxi
SUMMARY	xxiii
1. GİRİŞ	1
2. LİTERATÜR ÇALIŞMASI	5
3. VERİTABANI VE KAVRAMLAR.....	9
3.1. NoSQL Nedir?.....	9
3.2. Büyük Veri Nedir?	10
3.3. Bulut Bilişim Nedir?	11
3.4. NoSQL Yapısı	12
3.4.1. Acid teoremi.....	12
3.4.1.1. Bölünmezlik	12
3.4.1.2. Tutarlılık.....	13
3.4.1.3. Yalıtım	13
3.4.1.4. Dayanıklılık.....	13
3.4.2. Base teoremi.....	14
3.4.2.1. Temelde kullanılabilir	15
3.4.2.2. Geçici durum	15
3.4.2.3. Sonunda Tutarlı	15
3.4.3. Cap teoremi	16
3.4.3.1. Tutarlılık.....	16
3.4.3.2. Erişebilirlik.....	17
3.4.3.3. Bölünebilme toleransı	17
3.5. NoSQL’de Ölçeklendirme.....	18
3.5.1. Parçalama	18
3.5.2. Bölümleme	19
3.5.3. Kümeleme	21
3.5.4. Replikasyon.....	21
3.6. Veritabanı Çeşitleri	22
3.6.1. Anahtar-Değer.....	22
3.6.2. Geniş-Sütun.....	22
3.6.3. Doküman.....	23
3.6.4. Grafik	23
3.6.5. Mekansal	24
3.6.6. Blokszincir.....	24
3.6.7. Bellek-İçi.....	25

3.6.8. Vektör.....	26
3.6.9. Nesne tabanlı	26
3.6.10. Zaman-Serisi	27
3.6.11. NewSQL	27
3.6.12. SQL	28
3.7. MongoDB	29
3.7.1. Mimari	29
3.7.2. Veri modeli.....	30
3.7.3. Sorgu modeli	30
3.7.4. Çoğaltma	31
3.7.5. Bölümleme	31
3.8. Couchbase.....	32
3.8.1. Mimari	32
3.8.2. Veri modeli.....	34
3.8.3. Hata toleransı.....	35
3.9. Cassandra.....	36
3.9.1. Mimari	36
3.9.2. Sıkıştırma	38
3.9.3. Veri modeli.....	39
3.9.4. Hata toleransı.....	40
3.10. HBase	41
3.10.1. Mimari	41
3.10.2. Veri modeli.....	42
3.10.3. Yazma mekanizması	43
3.10.4. Okuma mekanizması	44
3.10.5. Sıkıştırma	44
4. TEST ORTAMI VE KONFIGÜRASYON	47
4.1. Küme Özelliği	47
4.2. YCSB.....	47
4.3. MongoDB Konfigürasyonu	49
4.4. Couchbase Konfigürasyonu.....	50
4.5. Cassandra Konfigürasyonu.....	51
4.6. HBase Konfigürasyonu	52
5. ARAŞTIRMA BULGULARI	55
5.1. İş Yüğü A	56
5.1.1. Test – 1	56
5.1.2. Test – 2	59
5.2. İş Yüğü B.....	61
5.2.1. Test – 1	61
5.2.2. Test – 2	63
5.3. İş Yüğü C.....	66
5.3.1. Test – 1	66
5.3.2. Test – 2	68
5.4. İş Yüğü D	71
5.4.1. Test – 1	71
5.4.2. Test – 2	73
5.5. İş Yüğü E.....	76
5.5.1. Test – 1	76
5.5.2. Test – 2	79
5.6. İş Yüğü F	81

5.6.1. Test – 1.....	81
5.6.2. Test – 2.....	84
5.7. İş Yükleri Ortalaması	87
5.7.1. Cassandra İş Yükleri Ortalaması.....	87
5.7.2. Couchbase İş Yükleri Ortalaması	88
5.7.3. MongoDB İş Yükleri Ortalaması.....	89
5.7.4. HBase İş Yükleri Ortalaması	90
6. SONUÇ VE ÖNERİLER.....	95
KAYNAKÇA	99
ÖZGEÇMİŞ.....	103





KISALTMALAR

ACID	: Atomicity, Consistency, Isolation, Durability
BASE	: Basic Availability, Soft State, Eventually Consistency
CAP	: Consistency, Availability, Partition Tolerance
CBS	: Coğrafi Bölge Sistemi
NYP	: Nesne Yönelimli Programlama
SQL	: Structured Query Language
BSON	: Binary JSON
JSON	: JavaScript Object Notation
N1QL	: SQL for JSON
DML	: Data Manipulation Language
HDFS	: Hadoop Distributed File System
CDC	: Change Data Capture
CQL	: Cassandra Query Language
SEDA	: Staged Event-Driven Architecture
SSTable	: Sorted String Table
G/Ç	: Giriş-Çıkış



SİMGELER

K	: Bin [Birim]
s	: Saniye [Birim]
GB	: Gigabyte [Birim]





TABLO LİSTESİ

Sayfa

Tablo 4.1. Test kapsamındaki sunucu donanım özelliği.	47
Tablo 5.1. Puanlama Tablosu - 1.....	91
Tablo 5.2. Puanlama Tablosu - 2	91
Tablo 5.3. Puanlama Tablosu - 3.....	91
Tablo 5.4. Puanlama Tablosu - 4.....	91
Tablo 5.5. Puanlama Tablosu - 5.....	91
Tablo 5.6. Puanlama Tablosu - 6.....	92
Tablo 5.7. Kullanım Senaryosu	93



NOSQL VERİTABANI SİSTEMLERİNİN PERFORMANS KARŞILAŞTIRILMASI VE ANALİZİ

ÖZET

Günümüzde veri miktarlarının hızla artması, geleneksel ilişkisel veritabanlarının performans ve ölçeklenebilirlik konusunda bazı zorluklarla yüzleşmesine neden olmuştur. Bu sorunların üstesinden gelmek amacıyla NoSQL veritabanı sistemleri yaygın olarak benimsenmiştir. Ancak, farklı NoSQL veritabanı sistemlerinin performansı, kullanım senaryolarına, veri modeline ve dağıtım mimarisine bağlı olarak büyük ölçüde değişiklik gösterebilmektedir.

"NoSQL Veritabanı Sistemlerinin Performans Karşılaştırması ve Analizi" başlıklı bu çalışma, çeşitli NoSQL veritabanı sistemlerinin belirli kullanım senaryolarında gösterdiği donanım performansını değerlendirmeyi amaçlamaktadır. Bu çalışma, karşılaştırmalı bir perspektif sunarak, kurum ve kuruluşların veritabanı altyapısı seçiminde daha bilinçli kararlar almasına yardımcı olmayı hedeflemektedir. Çalışmada, piyasada yaygın olarak kullanılan Cassandra, MongoDB, Couchbase ve HBase veritabanı sistemlerinin aynı donanım üzerinde gösterdiği performanslar, alan bilgisine dayalı yöntemlerle ele alınmıştır. Performans analizi için iş çıktısı oranı, gecikme süresi, işlemci kullanımı, bellek kullanımı ve iş parçacığı kullanımı gibi kritik metriklere odaklanılmıştır.

Analizlerin yapılabilmesi için her bir veritabanı sistemi minimum sistem gereksinimleri ile aynı donanım konfigürasyonu üzerinde kurulmuş ve test platformları oluşturulmuştur. Test platformları oluşturulduktan sonra her bir veritabanı üzerinde sanal sunucular oluşturularak belirlenen iş yükü senaryolarına göre test planları uygulanmıştır. İş yükü senaryoları; %50 okuma %50 güncelleme, %95 okuma %5 güncelleme, %100 okuma, %75 okuma %25 ekleme, %75 ekleme %25 güncelleme ve %50 okuma %50 okuma-güncelleme-yazma şeklinde belirlenmiştir.

Performans değerlendirme araçları kullanılarak gerçekleştirilen testler sonucunda elde edilen veriler bir tabloda kayıt altına alınmış ve her bir performans kriterine göre başarımların sıralaması yapılmıştır. Test sonuçlarına göre aynı donanım konfigürasyonunda iş çıktısı oranı, gecikme süresi, işlemci kullanımı ve bellek kullanımı açısından en iyi performansı Cassandra göstermiştir. Başarımların sıralamasında MongoDB ikinci, Couchbase üçüncü ve HBase dördüncü olmuştur. Bu çalışma, farklı veritabanı yönetim sistemlerinin performansını karşılaştırmalı olarak analiz ederek bilişim altyapısı seçiminde rehberlik sağlamaktadır. Elde edilen bulgular, kurum ve kuruluşların ihtiyaçlarına en uygun veritabanı sistemini seçmelerine yardımcı olacaktır.

Gelecek çalışmalarda, daha geniş veri setleri kullanılarak NoSQL veritabanlarının performansının daha kapsamlı bir şekilde incelenmesi önerilmektedir. Farklı iş yükü senaryoları ve kullanıcı davranışları dikkate alınarak performans testlerinin tekrarlanması, veritabanlarının çeşitli kullanım senaryolarında nasıl davrandığını anlamak açısından önemlidir. Ayrıca, farklı iş parçacığı sayıları ve paralel işlem

kapasitelerinin detaylı olarak analiz edilmesi, sistemlerin yüksek işlem yükleri altındaki performanslarının daha iyi anlaşılmasına yardımcı olabilir. Aynı testlerin farklı donanım konfigürasyonları üzerinde tekrarlanması, donanımın veritabanı performansı üzerindeki etkilerini belirlemek açısından faydalı olabilir. Veritabanı sistemlerinin güncel versiyonları ve yeni özellikleri dikkate alınarak performans testlerinin tekrarlanması, yazılım güncellemelerinin performans üzerindeki etkilerini değerlendirmek açısından önemlidir. Bu öneriler, NoSQL veritabanı sistemlerinin performansının daha kapsamlı ve detaylı bir şekilde incelenmesine olanak sağlayarak, gelecekte yapılacak çalışmalara önemli katkılar sunabilir.



COMPARISON AND ANALYSIS OF THE PERFORMANCE OF NOSQL DATABASE SYSTEMS

SUMMARY

The increasing data volumes generated by modern applications have posed significant challenges for traditional relational database management systems (RDBMS). While RDBMS have long served as a robust solution for structured data storage and retrieval, they often struggle with performance and scalability when confronted with the diverse and large-scale data requirements of today's digital environments. Traditional RDBMS architectures, with their rigid schema designs and reliance on complex join operations, are not always equipped to handle the flexible data models and real-time processing demands that are now commonplace. As the Internet of Things (IoT), social media platforms, and big data analytics continue to generate massive amounts of varied data, the limitations of RDBMS have become more evident, leading to the emergence of alternative database solutions.

NoSQL (Not Only SQL) databases have gained prominence as a flexible and scalable alternative to traditional RDBMS, addressing many of their inherent limitations. NoSQL databases offer a range of data models, such as key-value pairs, documents, wide-column stores, and graph structures, enabling them to manage unstructured and semi-structured data efficiently. This flexibility allows them to excel in applications that require rapid data ingestion, low-latency queries, and dynamic data structures. Industries such as e-commerce, social networking, and real-time analytics have widely adopted NoSQL databases due to their ability to manage large and varied datasets while maintaining high performance. These databases can scale horizontally across distributed systems, making them particularly suited for environments where the need for scalability and availability is paramount.

However, the performance characteristics of NoSQL databases can vary significantly based on usage scenarios, data models, and underlying distribution architectures. Understanding these differences is crucial for organizations aiming to select the right database system that aligns with their specific needs. This study, titled "Performance Comparison and Analysis of NoSQL Database Systems," provides a detailed evaluation of four widely used NoSQL databases: Cassandra, MongoDB, Couchbase, and HBase. The focus is on assessing their hardware performance across specific usage scenarios to help organizations make informed decisions regarding their database infrastructure. By analyzing key metrics such as throughput, latency, CPU usage, and memory usage, the study offers insights into how each database handles various workloads, aiding in the selection process for database solutions that meet specific operational requirements.

To ensure consistency and reliability in the results, each NoSQL database system was tested on identical hardware configurations, thereby eliminating hardware variability as a factor in performance differences. Virtual servers were deployed for each database, and the tests covered a range of scenarios, including balanced operations like 50% read and 50% update, predominantly read-heavy operations (95% read and 5%

update), and scenarios focused entirely on read operations (100% read). Additional tests included scenarios with high data insertion rates, such as 75% read with 25% insert and 75% insert with 25% update, as well as mixed read, update, and write operations. These scenarios represent common use cases found in various industries, from high-frequency transaction processing to real-time data analytics and content management systems.

The performance metrics measured in this study provide a comprehensive view of each database's capabilities. Throughput, measured in operations per second, indicates the ability of a database to manage large volumes of transactions efficiently, making it an essential metric for applications requiring high data flow. Latency, or response time, measures how quickly a database can process individual transactions, which is critical for user-facing applications where response speed directly impacts user experience. CPU and memory usage were also monitored to assess how each database utilizes system resources under different workloads, offering insights into the efficiency and scalability of each system. Additionally, thread usage was analyzed to determine how well each database can handle concurrent processing, which is particularly important for maintaining performance under high-load conditions.

The results of the comparative analysis revealed distinct strengths and weaknesses for each NoSQL database system. Cassandra emerged as a top performer in terms of throughput and maintaining low latency, particularly excelling in read-heavy and balanced scenarios. Its architecture, designed for high write throughput and horizontal scalability, enables it to manage large transaction volumes with minimal performance loss. However, Cassandra's memory usage was relatively high, suggesting that while it is effective in managing CPU resources, it may require further optimization for memory-intensive applications. This makes Cassandra a suitable choice for environments where high transaction throughput is crucial, but memory optimization can be managed through configuration tuning.

Couchbase, on the other hand, showed strong performance in scenarios that involved frequent data insertion, benefiting from its built-in caching mechanisms. These features allow Couchbase to achieve high-speed data ingestion and retrieval, making it effective for real-time applications that require rapid access to newly stored data. However, Couchbase exhibited variability in memory consumption across different scenarios, which indicates a need for careful management of resource allocation to maintain consistent performance. This suggests that Couchbase is particularly well-suited for environments where balanced read-write operations and quick data access are needed, though it may require additional fine-tuning for optimal resource use.

MongoDB demonstrated competitive performance, especially in mixed workload environments that involved a balance of read, update, and write operations. Its document-oriented model facilitates efficient querying and indexing, making it ideal for applications with dynamic data structures and varying query patterns. MongoDB's stability in CPU and memory usage across different workloads highlights its reliability in maintaining consistent performance. However, it faced challenges with latency in specific scenarios, particularly those with very high read or write demands, suggesting that while MongoDB is versatile, it may require performance optimization under extreme conditions.

HBase, while generally outperformed by the other systems in terms of throughput and latency, proved effective for specific use cases like batch processing and large-scale data analytics. Its strengths lie in handling extensive datasets with a need for consistent

data storage and retrieval over time, making it suitable for analytical applications that prioritize data consistency over speed. HBase's performance in environments where high volumes of data are processed over longer periods makes it a potential choice for organizations with specialized analytical needs, despite its limitations in scenarios that require low latency and high transaction throughput.

These findings underscore the importance of selecting a database system that aligns with the specific characteristics of the workload and application requirements. For applications that prioritize high write throughput and scalability, Cassandra may offer the most advantages. In contrast, MongoDB's querying capabilities make it suitable for read-heavy or dynamic data environments. Couchbase's efficiency in managing quick data ingestion and retrieval positions it as a good choice for balanced operations, while HBase is best suited for environments where batch processing and large-scale data storage are key considerations. Each database system has unique strengths that can be leveraged based on the specific needs of an organization.

The study also highlights the potential for optimizing infrastructure costs by understanding the resource consumption patterns of each database system. Databases with efficient CPU and memory usage, such as Cassandra, can reduce operational expenses while maintaining high performance levels. Fine-tuning configurations and indexing strategies based on the workload can further enhance performance, ensuring that the database infrastructure is optimized for the demands of the application. For organizations looking to maximize their investment in NoSQL technologies, such adjustments can provide significant long-term benefits in terms of both performance and cost efficiency.

Future research could expand this study by testing these databases with larger datasets and exploring the effects of hardware variations on performance outcomes. Including newer versions of these databases, as well as traditional RDBMS for comparison, could provide a more comprehensive view of the trade-offs between different database models. Such research would contribute to a deeper understanding of how NoSQL databases evolve over time and how they can be best applied in varying operational contexts. This study serves as a foundation for making informed decisions about database selection, helping organizations align their data management strategies with the specific performance requirements of their applications.

The detailed analysis of Cassandra, MongoDB, Couchbase, and HBase provides practical guidance for organizations in choosing the right NoSQL database system. By understanding the specific capabilities of each database, organizations can make more strategic decisions about their infrastructure, ensuring that their database solutions are optimized for the unique demands of their data-driven applications. This approach not only improves the performance of current systems but also allows organizations to adapt to future data growth and technological advancements.



1. GİRİŞ

Veri yönetimi, günümüzün dijital dünyasında kritik bir öneme sahiptir. Teknolojinin hızla gelişmesi, sosyal medyanın yaygınlaşması, akıllı telefonların yaygın kullanımı ve log verilerinin artmasıyla birlikte veri miktarı eksponansiyel olarak artmaktadır. Bu veri patlaması, büyük veri kavramının ortaya çıkmasına ve verilerin daha etkin bir şekilde işlenip yönetilmesi gerekliliğine neden olmuştur. Büyük veri, hacim, hız ve çeşitlilik (3V) özellikleriyle tanımlanır ve bu özellikler, geleneksel veri yönetim sistemleri tarafından yeterince karşılanamamaktadır.

Geleneksel ilişkisel veritabanları, yapılandırılmış veri yönetiminde uzun yıllardır standart olarak kabul edilmektedir. Ancak, bu sistemler büyük veri ile başa çıkmakta sınırlı kalmaktadır. İlişkisel veritabanları, verilerin önceden tanımlanmış şemalar içinde depolanmasını gerektirir ve bu, dinamik ve hızlı değişen veri ortamlarında esneklik sorunları yaratır. Bu noktada, NoSQL veritabanları, geleneksel ilişkisel veritabanlarının sınırlamalarını aşmak için geliştirilmiştir.

NoSQL veritabanları, yapılandırılmamış veya yarı yapılandırılmış veri formatlarında veri saklama yeteneğine sahiptir. Bu esneklik, metinler, resimler, videolar ve sensör verileri gibi çeşitli veri türlerini işlemek için idealdir. NoSQL veritabanları, dört ana kategoriye ayrılabilir: anahtar-değer tabanlı, belge tabanlı, sütun tabanlı ve grafik tabanlı veritabanları. Her bir kategori, belirli veri yönetimi ihtiyaçlarına uygun çözümler sunar. Örneğin, belge tabanlı veritabanları, JSON veya BSON formatında verileri depolayarak esnek ve dinamik veri yapıları sağlar.

Büyük veri, yüksek hacimli verilerin hızlı bir şekilde işlenmesini ve analiz edilmesini gerektirir. NoSQL veritabanları, yatay ölçeklenebilirlik özellikleri sayesinde bu gereksinimi karşılayabilir. Yatay ölçeklenebilirlik, veritabanlarının ek sunucular eklenerek genişletilmesini mümkün kılar, bu da performansın ve depolama kapasitesinin artırılmasını sağlar. Bu özellik, büyük veri ortamlarında NoSQL veritabanlarını tercih edilir kılar. Literatürde NoSQL veritabanlarının performansını karşılaştıran birçok çalışma bulunmaktadır. Ancak, yapılan çalışmalar incelendiğinde, NoSQL veritabanlarının performanslarının karşılaştırılması ve değerlendirilmesi

noktasında kaynak eksikliği olduğu ve mevcut çalışmalarda Türkçe kaynakların azlığı gözlemlenmektedir.

Bu çalışma ile NoSQL veritabanlarının performanslarının farklı iş yükleri ve veri türleri altında ölçülerek, birbirleri arasındaki farklılıkların ortaya konulması amaçlanmaktadır. Ayrıca, bu çalışmanın sonuçlarının kurum ve kuruluşların veri yönetimi stratejilerini belirlerken bilinçli kararlar almalarına yardımcı olması ve gelecekte yapılacak çalışmalar için yeni bir Türkçe kaynak oluşturulması hedeflenmektedir.

Çalışmada, MongoDB, Apache Cassandra ve HBase gibi popüler NoSQL veritabanları üzerinde çeşitli benchmark testleri gerçekleştirilecektir. Bu testler, veritabanlarının okuma, yazma, güncelleme ve karmaşık sorgulama performanslarını değerlendirmeyi amaçlamaktadır. Farklı veri yapıları ve iş yükleri altında gerçekleştirilecek bu testler, NoSQL veritabanlarının ölçeklenebilirlik, dağıtık mimari performansı ve veri modelleme yeteneklerini ortaya koyacaktır.

Ayrıca, bu çalışma, NoSQL veritabanlarının güvenlik, veri bütünlüğü ve uyumluluk gibi kritik konulardaki performanslarını da inceleyecektir. Enerji tüketimi ve verimlilik gibi konular da değerlendirilecek ve NoSQL veritabanlarının sürdürülebilir veri yönetimi açısından avantajları ve dezavantajları belirlenecektir.

Birçok çalışma sentetik iş yükleri ve laboratuvar ortamlarında gerçekleştirilmiştir. Gerçek dünya uygulamalarındaki performansın ve kullanım durumlarının daha ayrıntılı incelenmesi gerekmektedir. Bu, NoSQL veritabanlarının pratikteki performansını daha doğru bir şekilde yansıtacaktır.

Literatürdeki birçok çalışma genellikle yapılandırılmış veya yarı yapılandırılmış veriler üzerinde durmaktadır. Ancak, farklı veri türleri ve karmaşık veri yapılarının performans üzerindeki etkilerinin daha fazla araştırılması gerekmektedir. Bu, NoSQL veritabanlarının çeşitli veri türleriyle nasıl başa çıktığını anlamamıza yardımcı olacaktır.

Performans değerlendirmelerinde güvenlik, veri bütünlüğü ve uyumluluk gibi kritik konular genellikle göz ardı edilmektedir. Bu alanlarda daha fazla çalışma yapılması gerekmektedir. Güvenlik ve uyumluluk konularının dikkate alınması, NoSQL veritabanlarının daha güvenilir ve uyumlu hale gelmesini sağlayacaktır.

NoSQL veritabanlarının enerji tüketimi ve verimliliği hakkında daha fazla araştırma yapılması gerekmektedir. Özellikle büyük veri merkezlerinde enerji maliyetlerinin önemli olduğu durumlarda bu konular büyük önem taşımaktadır. Enerji verimliliğinin incelenmesi, daha sürdürülebilir veri yönetim sistemlerinin geliştirilmesine katkı sağlayacaktır.

Bu çalışmada, performans değerlendirmesi kapsamında MongoDB, Apache Cassandra, Couchbase ve HBase veritabanları incelenecektir. Bu veritabanları, büyük veri işleme ve dağıtık sistemlerde yaygın olarak kullanılmaları nedeniyle seçilmiştir.

Çalışmanın amacı doğrultusunda MongoDB, Apache Cassandra, Couchbase ve HBase veritabanlarının iş çıktısı, gecikme süresi, işlemci kullanımı ve bellek kullanımı performanslarının ölçülmesi ve karşılaştırılması amaçlanmaktadır. Bu doğrultuda, iş çıktısı, gecikme süresi, işlemci kullanımı ve bellek kullanımı performans kriterleri olarak belirlenmiştir.

Seçilen performans metriklerinin değerlendirilmesi amacıyla MongoDB, Apache Cassandra, Couchbase ve HBase veritabanlarının test edileceği ortamların oluşturulması ve performans ölçüm araçlarının belirlenmesi gerekmektedir. Test ortamlarının oluşturulması sürecinde, veritabanlarının donanım ihtiyaçları dikkate alınarak kurulumlar yapılmıştır. Performans ölçüm araçları ise belirlenen metriklerin değerlendirilmesi ve sistem üzerindeki yükün en aza indirilmesi kriterlerine göre seçilmiştir.

Altı ana bölümden oluşan bu çalışmanın ilk bölümü olan Giriş bölümünde, çalışmaya temel olan problem, çalışmanın yapılma amacı, çalışmanın kapsamı, çalışmada kullanılan yöntem ve çalışmanın genel yapısı tanıtılmaktadır. Çalışmanın ikinci bölümü olan Literatür Çalışması bölümünde, çalışma öncesi bakış açısı kazanmak ve teorik bilgi edinmek amacıyla MongoDB, Couchbase, Cassandra ve HBase veritabanları, bu veritabanlarının mimarileri ve performans analizleri kapsamında daha önce yapılmış ve çalışma konusuna katkı sağlayabilecek çalışmalar incelenmektedir. Çalışmanın üçüncü bölümü olan Veritabanı ve Kavramlar bölümünde, Literatür Çalışmasından elde edilen bakış açısı ve teorik bilgi birikimi doğrultusunda çalışmanın daha iyi anlaşılabilmesi için NoSQL veritabanları, büyük veri, bulut bilişim ve NoSQL yapısı hakkında bilinmesi gereken temel kavramlar ele alınmaktadır. Bu bölümde ayrıca Acid, Base ve Cap teoremleri, NoSQL’de ölçeklendirme ve çeşitli veritabanı

türleri detaylandırılmaktadır. Çalışmanın dördüncü bölümü olan Test Ortamı ve Konfigürasyon bölümünde, performans analizi için izlenen metodolojik yaklaşım, test ortamlarının hazırlanışı ve kullanılan araçlar detaylı olarak açıklanmaktadır. Bu bölümde ayrıca, YCSB (Yahoo Cloud Serving Benchmark) aracının kullanımı ve MongoDB, Couchbase, Cassandra ve HBase veritabanlarının konfigürasyonları anlatılmaktadır. Çalışmanın beşinci bölümü olan Araştırma Bulguları bölümünde, farklı senaryolar üzerinden yapılan testlerden elde edilen verimlilik, gecikme, CPU kullanımı bellek kullanımı ve iş parçacığı kullanımı verilerinin analizi ve veritabanlarının karşılaştırılması ele alınmıştır. Çalışmanın altıncı bölümü olan Sonuç ve Öneriler bölümünde bulguların değerlendirmesi, mevcut çalışmalarla karşılaştırma ve gelecekteki çalışmalara yönelik öneriler sunulmaktadır.



2. LİTERATÜR ÇALIŞMASI

Birçok çalışma, NoSQL veritabanlarının performansını karşılaştırmış ve analiz etmiştir. Dumanlı, (2020) çalışmasında, ilişkisel ve NoSQL veritabanı yönetim sistemlerini karşılaştırmış ve iş yüküne bağlı olarak, NoSQL sistemlerinin ölçeklenebilirlik ve dağıtılmış performans avantajları sunduğunu bulmuştur. Bununla birlikte, ilişkisel sistemlerin ACID uyumluluğu ve yapılandırılmış veri modellemesinde önemli güçlü yönler gösterdiği vurgulanmıştır. Her iki tür veritabanı yönetim sisteminin de kullanım senaryolarına ve veri gereksinimlerine bağlı olarak farklı avantajlar ve dezavantajlar sunduğu belirtilmiştir.

Hammood, (2021) çalışmasında, MongoDB, Apache HBase ve Apache Cassandra gibi NoSQL veritabanı sistemlerini karşılaştırmıştır. Çalışma, bu üç sistem arasındaki performans, ölçeklenebilirlik, veri modeli ve kullanım kolaylığı açısından farklılıkları ortaya koymuştur. Sonuçlar, her bir veritabanı sisteminin uygulama gereksinimlerine bağlı olarak benzersiz avantajlar sunduğunu vurgulamış ve uygulama senaryosuna dayalı olarak uygun sistemin seçilmesinin önemini ortaya koymuştur.

Seçgin, (2019) çalışmasında, ilişkisel ve NoSQL veritabanı sistemlerinin performansını karşılaştırmıştır. Çalışmanın bulgularına göre, ilişkisel veritabanı sistemleri genel olarak yapılandırılmış veri modellemesi ve ACID uyumluluğunda avantajlara sahipken, ölçeklenebilirlikte sınırlı olabilmektedir. Öte yandan, NoSQL sistemleri, ölçeklenebilirlikleri ve dağıtılmış veritabanı mimarileri nedeniyle büyük veri işleme ve yüksek performanslı uygulamalar için daha uygun olabilmektedir. Sonuçlar, veritabanı yönetim kararlarının uygulama gereksinimleri ve performans önceliklerine dayalı olarak alınması gerektiğini vurgulamaktadır.

Li & Manoharan, (2013) çalışmasında, SQL ve NoSQL veritabanlarının performansını karşılaştırmış ve MongoDB, Cassandra ve HBase üzerinde yapılan testlerin, yüksek okuma ve yazma iş yükleri altında NoSQL veritabanlarının SQL veritabanlarına göre daha iyi performans gösterdiğini ortaya koymuştur. Özellikle Cassandra'nın yüksek yazma performansı dikkat çekmiştir.

Cooper ve ark., (2010) çalışmasında, NoSQL veritabanlarının performansını YCSB (Yahoo! Cloud Serving Benchmark) kullanarak değerlendirmiştir. Çalışmada, MongoDB, Cassandra ve HBase'in çeşitli iş yükü senaryoları altında performansını karşılaştırmışlardır. Bu senaryolar arasında %50 okuma ve %50 güncelleme, %95 okuma ve %5 güncelleme, %100 okuma, %75 okuma ve %25 ekleme, %75 ekleme ve %25 güncelleme, ve %50 okuma ve %50 okuma-değiştirme-yazma gibi senaryolar bulunmaktadır. Farklı veritabanlarının çeşitli iş yükleri altında nasıl performans gösterdiğini ayrıntılı olarak analiz etmişlerdir.

Moniruzzaman ve Hossain, (2013) çalışmalarında, NoSQL veritabanlarının büyük veri analitiği bağlamında nasıl kullanıldığını ve bu sistemlerin hangi iş yükleri için en uygun olduğunu analiz etmişlerdir. Çalışma, MongoDB'nin esnek veri modelleme yeteneklerini, Couchbase'in düşük gecikmeli veri erişimini, Cassandra'nın yüksek yazma performansını ve HBase'in büyük veri analitiği için uygunluğunu ayrıntılı olarak açıklamaktadır.

(Pokorny, 2013) çalışmasında, NoSQL veritabanlarının web ortamlarında ölçeklenebilirliği artırmadaki rolünü araştırmış ve MongoDB, Couchbase, Cassandra ve HBase gibi veritabanlarının performansını karşılaştırarak farklı kullanım senaryoları için en uygun çözümleri belirlemiştir.

(Stonebraker, 2010) çalışmasında, SQL ve NoSQL veritabanlarının performansını karşılaştırmış ve farklı veri işleme gereksinimlerine dayalı olarak hangi tür veritabanının daha uygun olduğunu tartışmıştır. Bu çalışmada, büyük ölçekli veri işleme senaryolarında NoSQL veritabanlarının avantajları vurgulanmıştır.

(Grolinger ve ark., 2013) çalışmasında, bulut ortamlarında veri yönetimi için NoSQL ve NewSQL veritabanlarının performansını incelemiş, ölçeklenebilirliklerini ve performans avantajlarını karşılaştırmıştır. Bu çalışmada, NoSQL veritabanlarının büyük veri işleme konusundaki yetenekleri ve performans avantajları detaylandırılmıştır.

Birçok çalışma sentetik iş yükleri ve laboratuvar ortamlarında gerçekleştirilmiştir. Gerçek dünya uygulamalarındaki performansın ve kullanım durumlarının daha ayrıntılı incelenmesi gerekmektedir. Bu, NoSQL veritabanlarının pratikteki performansını daha doğru bir şekilde yansıtacaktır.

Literatürdeki birçok çalışma genellikle yapılandırılmış veya yarı yapılandırılmış veriler üzerinde durmaktadır. Ancak, farklı veri türleri ve karmaşık veri yapılarının performans üzerindeki etkilerinin daha fazla araştırılması gerekmektedir. Bu, NoSQL veritabanlarının çeşitli veri türleriyle nasıl başa çıktığını anlamamıza yardımcı olacaktır.

Performans değerlendirmelerinde güvenlik, veri bütünlüğü ve uyumluluk gibi kritik konular genellikle göz ardı edilmektedir. Bu alanlarda daha fazla çalışma yapılması gerekmektedir. Güvenlik ve uyumluluk konularının dikkate alınması, NoSQL veritabanlarının daha güvenilir ve uyumlu hale gelmesini sağlayacaktır.

NoSQL veritabanlarının enerji tüketimi ve verimliliği hakkında daha fazla araştırma yapılması gerekmektedir. Özellikle büyük veri merkezlerinde enerji maliyetlerinin önemli olduğu durumlarda bu konular büyük önem taşımaktadır. Enerji verimliliğinin incelenmesi, daha sürdürülebilir veri yönetim sistemlerinin geliştirilmesine katkı sağlayacaktır.



3. VERİTABANI VE KAVRAMLAR

3.1. NoSQL Nedir?

NoSQL, geleneksel SQL veritabanı yönetim sistemlerine alternatif olarak geliştirilen bir veritabanı yaklaşımıdır. NoSQL veritabanları, yüksek ölçeklenebilirlik, yapısal olarak esneklik, ağ sorunlarının üstesinden gelebilme ve farklı veri türlerini (örneğin belge, anahtar-değer, geniş-sütun ve grafik) depolamak ve sorgulamak için kullanılmaktadır. Bazı NoSQL veritabanlarındaki öncelikli hedef yüksek erişebilirlik ve bölünebilme toleransı iken, bazı NoSQL veritabanlarında tutarlılık ve bölünebilme toleransı öne çıkmaktadır. İlerleyen süreçte ise tutarlılık, yüksek erişebilirlik ve bölünebilme toleransı hedeflenmektedir.

NoSQL veritabanlarının en çok kullanıldığı alanların başında büyük veri sistemleri gelmektedir. Bu durum, NoSQL veritabanlarının büyük veri sistemlerinde sıklıkla kullanılmasının temel nedenlerindendir: Ölçeklenebilir olmaları, büyük hacimli yapıları destekleyebilmeleri ve çeşitli veri türlerini (yapısal, yarı yapısal ve yapısal olmayan) aynı veritabanı içinde barındırmalarıdır.

NoSQL veritabanlarının tercih edilme sebeplerinden bir diğeri ise ölçeklenebilirlikleridir. İlişkisel veritabanı yönetim sistemlerinde performans artışı genelde merkezi bir cihaza bağlı olarak dikey ölçeklendirme ile sağlanırken (örneğin tek bilgisayarda yüksek bellek ve işlemci kullanımı), NoSQL veritabanları yatay ölçeklendirmeye izin vermektedir. Bu da gerektiği ya da gerekeceği kadar mimariye yatırım yapma olanağı sağlamaktadır.

NoSQL veritabanlarının ortaya çıkmasının bir diğer nedeni veri çeşitliliğidir. Geleneksel ilişkisel veritabanı yönetim sistemleri yapısal bir mimari olma zorunluluğu taşıırken, NoSQL veritabanları yarı yapısal veya yapısal olmayan bir mimaride verileri saklayabilme özelliğine sahiptir. Bu da kullanıcılar için tablolar arasında zorunlu ilişki gerektirmeyen, farklı yapıdaki verileri işleyebilme esnekliği sağlamaktadır. Bu da NoSQL veritabanlarının kullanımını gün geçtikçe artırmaktadır (Brown ve Johnson, 2020).

3.2. Büyük Veri Nedir?

Günümüzde veri yönetimi, büyük veri kavramıyla birlikte önemli bir dönüşüm yaşamıştır. Özellikle sosyal medyanın yaygın kullanımı, akıllı telefonlar, sensör verileri ve internet üzerindeki etkileşimler, veri üretimini hızlandırmıştır. Bu veriler kimi zaman yapısal, kimi zaman yarı-yapısal ve kimi zaman ise yapısal olmayan şekilde oluşmaktadır. Büyük veri formundaki yapılara bakacak olursak;

Yapısal Veri: Verinin formatı düzenli ve sıralı yapıya sahiptir. Bu tür veriler, ilişkisel veritabanlarında tablo biçiminde depolanabilir ve genellikle sütunlar ve satırlar şeklinde saklanabilir. Örnek olarak, adres, telefon, eposta adresleri gösterilebilir.

Yarı Yapısal Veri: Veri formatı yarı düzenli ve sıralı şekildedir. Sabit bir yapı mimarisi yoktur ve tüm veriler aynı satır, sütun dizgisinde değildir. Geleneksel yöntemlerdeki gibi biçimlendirilemezler. Buna karşın işlenmesine yardımcı olacak şekilde yapısal öğeleri içerirler.

Yapısal Olmayan Veri: Veri formatı hiçbir yapıya sahip değildir. İçerikleri düzensiz ve yapısal değildir. Bu veri türüne örnek olarak ses, video, fotoğraf gibi çeşitli veriler verilebilir.

2001 yılında, Doug Laney, büyük verinin tanımını İngilizce baş harflerini kullanarak hacim, hız ve çeşitlilik şeklinde 3V(volume, velocity, variety) denilen üç bileşen ile ifade etmiştir.



Şekil 3.1. Büyük Veri 3V Modeli.

Bazı kaynaklar yaklaşımı 4V-5V olarak tanımlarken hacim, hız, çeşitlilik ve doğruluk, değer olarak tanımlarken kimi kaynaklarda büyük veri 17V şeklinde tanımlanmaktadır (Laney, 2001).

3.3. Bulut Bilişim Nedir?

Bilgi işlem hizmetlerinin (örneğin sunucu, depolama, veritabanı, analitik araçlar) internet üzerinden sağlandığı ve yönetilmesi anlamına gelmektedir. İşletmeler ve bireyler, gereksinimlerine uygun kaynakları kullanarak fiziksel sunuculara ve altyapılara yatırım yapmadan işlerini yürütebilmektedirler. Bu yaklaşım, maliyetlerin optimize edilmesini sağlamakta ve işletmelerin esnek bir şekilde büyümesine imkân tanımaktadır. Kullanıcılar, ihtiyaç duydukları bilgisayar kaynaklarını talep edebilmekte, kullanabilmekte ve bunlara internet üzerinden erişebilmektedirler.

Bulut bilişim, geleneksel bilişim modellerine kıyasla bir dizi avantaj sunar:

Ölçeklenebilirlik: Kullanıcılar, ihtiyaçlarına göre kaynakları kolayca ölçeklendirebilirler. Böylece, iş yükü arttıkça veya azaldıkça uygun kaynakları kullanarak maliyetleri optimize edebilirler.

Pay-As-You-Go Modeli: Bulut bilişim genellikle bir abonelik modeli üzerine kuruludur. Kullanıcılar, ihtiyaçlarına göre kaynakları kiralarlar ve sadece kullandıkları kaynaklar için ödeme yaparlar. Bu, maliyetleri optimize etmeye ve kaynakların verimli bir şekilde kullanılmasına olanak tanır.

Erişilebilirlik: Bulut bilişim, internet bağlantısı olan herhangi bir cihazdan erişilebilir. Bu, kullanıcıların ve işletmelerin coğrafi sınırlamalardan bağımsız olarak kaynaklara erişmelerini sağlar.

Yedekleme ve Kurtarma: Bulut sağlayıcıları, verilerin yedeklenmesi ve felaket kurtarma için genellikle kapsamlı hizmetler sunarlar. Bu, veri kaybını en aza indirir ve iş sürekliliğini sağlar.

Esneklik ve Çeşitlilik: Bulut bilişim, farklı hizmet modelleri (altyapı, platform ve yazılım hizmetleri) ve dağıtım modelleri (genel, özel, topluluk ve hibrit bulutlar) sunar. Bu, kullanıcıların ihtiyaçlarına ve tercihlerine uygun çözümler seçmelerine olanak tanır (Armbrust ve ark., 2010).

3.4. NoSQL Yapısı

3.4.1. Acid teoremi

ACID (Atomicity, Consistency, Isolation, Durability) teoremi, modern veritabanı yönetiminde temel bir rol oynayan ve veri işlemlerinin güvenilirliğini sağlamak için kritik öneme sahip bir set ilkedir. Bölünmezlik, bir işlemin ya tamamıyla gerçekleşmesini ya da hiç gerçekleşmemesini garanti etmektedir. Bu, işlem sırasında herhangi bir aşamada hata olması durumunda, işlemin geri alınacağı anlamına gelmektedir. Tutarlılık, veritabanının belirli kısıtlamalara, ilişkilere ve kurallara uygun olarak işlem görmesini sağlamaktadır. Yalıtım, eş zamanlı işlemlerin birbirinden bağımsız olarak çalışmasını ve bir işlemin diğer işlemleri etkilememesini sağlamaktadır. Son olarak, Dayanıklılık, bir işlemin tamamlanmasının ardından yapılan değişikliklerin kalıcı olmasını ve sistem çökmesi durumunda dahi verinin korunmasını sağlamaktadır.

ACID teoremi, bu dört temel ilkenin bir araya gelmesiyle, veri tabanı işlemlerinin güvenilirliğini ve tutarlılığını güvence altına almaktadır. Bu ilkelere dayanan sistemler, kritik uygulamalarda veri bütünlüğünü sağlamak için tercih edilmektedir. Veritabanı işlemlerinin atomik olması, tutarlı olması, izole olması ve sağlam olması, modern veri yönetimi gereksinimlerine uygun esnek ve performanslı bir altyapı sunmaktadır. Bu sayede, işlemlerin güvenilir bir şekilde gerçekleştirilmesi ve verinin korunması mümkün olmakta, bu da ACID teoreminin önemini ve etkinliğini vurgulamaktadır (Silberschatz ve ark., 2020).

3.4.1.1. Bölünmezlik

Bir işlem, işlemin tamamının ya tamamıyla gerçekleştiği ya da hiç gerçekleşmediği şeklinde tanımlanmaktadır. Bu özellik, işlemlerin parçalanamaz bir bütün olarak ele alınmasını sağlamakta, yani işlem içindeki her adımın başarıyla tamamlanması gerekliliğini getirmektedir. Bir işlem başarıyla tamamlandığında, işlem sırasında yapılan tüm değişiklikler kalıcı hale gelmekte ve veritabanı stabil bir durumda kalmaktadır. Ancak, işlem sırasında herhangi bir aşamada hata oluşması veya kesinti meydana gelmesi durumunda, işlem geri alınmakta veya iptal edilmektedir. Bu durum, işlemin veritabanı üzerindeki bütünlüğünü korumak ve veri bütünlüğü kısıtlamalarına uygunluğunu sağlamak için önemli olmaktadır.

İşlem tamamlanmadan önce herhangi bir hata durumunda, veritabanı işlem öncesi durumuna geri dönmekte ve kullanıcıya işlem tamamlanamadı bilgisi verilmektedir. Bu şekilde, işlemlerin güvenilirliği ve veritabanı bütünlüğü korunmuş olmakta ve kullanıcılar için tutarlı bir deneyim sağlanmaktadır (Silberschatz ve ark., 2020).

3.4.1.2. Tutarlılık

Bir işlem, veritabanını bir geçerli durumdan diğerine geçirirken, veritabanının belirli kurallara (kısıtlamalar, ilişkiler vb.) uygunluğunu korumaktadır. Bu durum, veritabanı yönetim sistemlerinde önemli bir kavram olan veri bütünlüğünü sağlamak için gerekli olmaktadır. İşlem, veritabanında yapılan herhangi bir değişikliğin, tanımlanan kısıtlamalara ve ilişkilere uygun olarak gerçekleştirilmesini sağlamaktadır. Örneğin, bir işlem sonucunda bir tablo içindeki verilerde bir değişiklik yapıldığında, bu değişiklik ilgili referans anahtarları veya diğer ilişkilerle uyumlu olmalıdır. Böylece, veritabanı her zaman geçerli ve doğru verilere sahip olmakta ve bütünlük korunmuş olmaktadır. Veri bütünlüğü, veritabanı uygulamalarında güvenilirliği ve tutarlılığı sağlamak için temel bir gereklilik olup, işlemlerin bu prensibe uygun olarak yönetilmesi, veritabanı sisteminin güvenilirliğini artırmaktadır (Silberschatz ve ark., 2020).

3.4.1.3. Yalıtım

Bir işlem, diğer işlemlerden bağımsız olarak gerçekleşmektedir ve bu özelliğiyle eş zamanlı işlemlerin etkileşimlerini kontrol altında tutmaktadır. Her işlem, kendi içinde tutarlı bir şekilde işlem yapmakta ve diğer işlemler üzerinde herhangi bir etkisi olmamaktadır. Bu, veritabanı yönetiminde önemli bir prensiptir çünkü eş zamanlı olarak çalışan birçok işlem, aynı veri kaynaklarını kullanabilmekte ve bu nedenle veritabanının bütünlüğünü riske atabilmektedir. Bir işlem, başlangıç ve bitiş arasında veritabanının geçerli bir durumda kalmasını sağlamalıdır. Bu şekilde, veritabanı her zaman geçerli bütünsel kısıtlamalara uygun bir durumda kalmakta ve kullanıcılar arasında tutarlı bir deneyim sunulmaktadır (Silberschatz ve ark., 2020).

3.4.1.4. Dayanıklılık

Dayanıklılık, veri yönetiminde kritik bir ilke olup, bir işlemin tamamlanmasının ardından yapılan değişikliklerin kalıcı olarak saklanmasını sağlamaktadır. Bu ilke, veri saklayıcıların (örneğin disk veya SSD) arızalanması, beklenmeyen sistem çökmeleri

veya güç kesintileri gibi beklenmedik durumlar karşısında dahi verinin korunmasını garanti etmektedir.

Veri yönetiminde dayanıklılık ilkesi, kullanıcılar ve uygulamalar için büyük önem taşımaktadır çünkü veri kaybını önlemekte ve veri bütünlüğünü sağlamaktadır. Bir işlemin tamamlanmasının ardından verinin kalıcı olarak korunması, veritabanı sistemlerinin güvenilirliğini ve dayanıklılığını artırmaktadır. Bu durum, kullanıcıların veriye güvenle erişebilmesini ve işlemlerini güvenilir bir şekilde tamamlayabilmesini sağlamaktadır. Dayanıklılık ilkesi, modern veritabanı sistemlerinin temel prensiplerinden biri olarak, veri yönetiminde güvenilirliği ve sürekli erişilebilirliği sağlamak için kritik bir rol oynamaktadır (Silberschatz ve ark., 2020).

3.4.2. Base teoremi

BASE (Basically Available, Soft State, Eventually Consistent) prensibi, modern veritabanı yönetiminde önemli bir rol oynamaktadır. Bu prensip, anında tutarlılığın gerektiği durumlar yerine, kullanılabilirlik ve esneklik üzerine odaklanmaktadır. Dağıtık sistemlerde ağ gecikmeleri veya bağlantı sorunları gibi durumlar, bazen anında tutarlı veri erişimini engelleyebilmektedir. BASE prensibi, böyle durumlarda sistemin devamlılığını sağlamak amacıyla veri tutarlılığını zaman içinde garanti altına alma ilkesine dayanmaktadır.

BASE'in temel bileşenleri olan Temelde Kullanılabilir, Geçici Durum ve Sonunda Tutarlı, veritabanı yönetiminde esneklik ve performans arasında bir denge sağlamaktadır. Temelde Kullanılabilirlik, sistemdeki her veri erişiminin başarılı bir şekilde tamamlanmasını ve kullanıcılara yanıt verilmesini garanti etmektedir. Geçici Durum, verinin geçici olarak tutarsız olabileceği ve zamanla tutarlı hale geleceği anlamına gelmektedir. Sonunda Tutarlılık ise, veri kopyaları arasındaki farkların zamanla düzeleceği ve tutarlılığın sağlanacağı garantisini vermektedir.

Bu prensip, özellikle sosyal medya platformları, ağ tabanlı uygulamalar ve büyük ölçekli dağıtık sistemler gibi alanlarda kullanılmaktadır. Örneğin, sosyal medya platformları kullanıcı etkileşimleri veya içerik güncellemeleri için anında kesin veri tutarlılığına ihtiyaç duymamaktadır; bunun yerine, kullanılabilirlik ve performans üzerinde daha fazla vurgu yapmaktadır. BASE prensibi, veritabanı yönetiminde esneklik, kullanılabilirlik ve performansın dengeli bir şekilde sağlanmasına yardımcı olmaktadır. Bu sayede, modern uygulamaların gereksinimlerini karşılayacak şekilde veri yönetimi yapılandırılmakta ve optimize edilmektedir (Pritchett., 2008).

3.4.2.1. Temelde kullanılabilir

Temelde Kullanılabilirlik bileşeni, modern dağıtık sistemlerde kullanıcı erişimini sürekli olarak sağlamayı amaçlamaktadır. Bu prensip, sistemde herhangi bir aksama veya kesinti durumunda bile belirli temel hizmetlerin devam etmesini sağlamaktadır. Temelde Kullanılabilirlik ilkesi, sistemdeki diğer bileşenlerde veya hizmetlerde yaşanabilecek sorunlara rağmen belirli işlevlerin ve hizmetlerin kullanıcıya sunulmasını garanti etmektedir. Bu sayede, kullanıcıların sisteme sürekli olarak erişim sağlaması ve belirli temel işlemleri gerçekleştirmesi mümkün olmaktadır (Bondi., 2000).

3.4.2.2. Geçici durum

NoSQL veritabanlarının tasarımında önemli bir kavram olarak öne çıkmaktadır. Bu ilke, veritabanı sisteminin geçici bir tutarsızlık durumunu kabul edilebilir kılmaktadır. NoSQL veritabanları genellikle dağıtık ortamlarda çalıştıkları için, ağdaki gecikmeler, düğümlerdeki zaman uyumsuzlukları veya ağ kesintileri gibi durumlar söz konusu olabilmektedir. Bu durumlar, veritabanının farklı kopyaları arasında geçici olarak farklılık oluşturabilmekte ve kullanıcıların aynı anda farklı sonuçlar görmesine neden olabilmektedir.

Bu ilke, NoSQL sistemlerinin anlık konsensüs veya anlık tutarlılık sağlama zorunluluğunu kaldırmakta ve böyle durumlarda sistemdeki tutarsızlıkların geçici olarak kabul edilmesini öngörmektedir. Yani, kullanıcılar belirli bir sorgulama veya işlemde geçici olarak farklı sonuçlar görebilmekte, ancak sistem zamanla bu durumu düzeltecek ve veritabanının tüm kopyalarını yeniden tutarlı hale getirecektir. Bu süreç, veritabanı sisteminin yeniden stabil hale gelmesini sağlayacak düzelme mekanizmaları veya veri senkronizasyon yöntemleri tarafından yönetilmektedir.

Geçici durum ilkesi, NoSQL veritabanlarının ölçeklenebilirlik, performans ve kullanılabilirlik gibi avantajlarını korurken, aynı zamanda katı tutarlılık gereksinimlerinden esneklik sağlamaktadır. Bu sayede, büyük veri miktarlarının yönetimi ve yoğun kullanıcı trafiği altında veritabanı sisteminin sağlam bir şekilde işlemlerini mümkün kılmaktadır (Stonebraker ve ark., 2005).

3.4.2.3. Sonunda Tutarlı

NoSQL veritabanlarının dağıtık sistemlerde çalıştığı durumlarda tutarlılığı sağlamak için benimsenen bir yaklaşımı ifade etmektedir. NoSQL veritabanları, genellikle

büyük ölçekli ve dağıtık sistemlerde kullanıldığından, veri kopyaları arasında anlık konsensüs sağlamak zor olabilmektedir. Bu durumda, ağ gecikmeleri, düğüm hataları veya diğer sistemsel sorunlar nedeniyle veri kopyaları arasında geçici farklılıklar oluşmaktadır.

Sonunda tutarlı ilkesi, bu tür durumları kabul etmekte ve sistemin zamanla tutarlı hale gelmesini beklemektedir. Yani, bir kullanıcı veya uygulama belirli bir veriyi sorguladığında veya güncellediğinde, her zaman en güncel veya tamamen tutarlı bir sonuç alması garanti edilmemektedir. Ancak, sistem tasarımı ve veri senkronizasyon mekanizmaları sayesinde, veritabanının tüm kopyaları zamanla aynı duruma gelmekte ve tutarlılık sağlanmaktadır (Terry ve ark., 2013).

3.4.3. Cap teoremi

CAP teoremi, dağıtık veritabanı sistemlerinin temel kısıtlamalarını ve tasarım seçeneklerini belirleyen önemli bir teorik çerçevedir. Bu teorem, Tutarlılık, Erişilebilirlik ve Bölümleme Toleransı olmak üzere üç temel özellik arasında bir seçim zorunluluğunu açıklamaktadır.

Tutarlılık, bir veritabanı sistemi içindeki verilerin her zaman güncel ve tutarlı olmasını ifade etmektedir. Erişilebilirlik, sistemdeki her istemcinin her zaman cevap alabilmesini sağlamayı amaçlamaktadır. Bölümleme Toleransı ise, ağdaki kesintiler veya düğüm hataları gibi bölünmelerden sonra bile sistemdeki işlemlerin devam etmesini mümkün kılan bir özelliktir.

CAP teoremi, bu üç özelliğin bir dağıtık veritabanı sistemi içinde aynı anda sağlanamayacağını belirtmektedir. Yani, bir sistem tasarımı sırasında, Tutarlılık, Erişilebilirlik ve Bölümleme Toleransı arasında bir denge kurulması gerekmektedir. Örneğin, bir sistem yüksek Erişilebilirlik sağlamak için tutarlılıkta bazı esneklikler yapabilir veya Bölümleme Toleransı'nı artırmak için Tutarlılıktan ödün verebilir. Bu seçimler, genellikle sistem performansı, veri bütünlüğü ve kullanılabilirlik açısından dikkatli bir denge gerektirmektedir (Maçek & Sajjanhar, 2017).

3.4.3.1. Tutarlılık

NoSQL veritabanlarının dağıtık sistemlerde çalıştığı durumlarda tutarlılığı sağlamak için benimsenen bir yaklaşımı ifade etmektedir. NoSQL veritabanları, genellikle büyük ölçekli ve dağıtık sistemlerde kullanıldığından, veri kopyaları arasında anlık konsensüs sağlamak zor olabilmektedir. Bu durumda, ağ gecikmeleri, düğüm hataları

veya diğer sistemsel sorunlar nedeniyle veri kopyaları arasında geçici farklılıklar oluşmaktadır.

Tutarlılık, dağıtılmış veritabanı sistemlerindeki temel bir gerekliliktir ve her istemcinin sistemdeki verilerin güncel ve tutarlı olduğunu sağlamaktadır. Bu, veritabanı sisteminin tüm kullanıcıları için aynı anda güncel verilere erişim sağlaması gerektiği anlamına gelmektedir. Tutarlılık, özellikle aynı anda gerçekleştirilen çoklu işlemlerde veya paralel erişim durumlarında verilerin doğru ve beklendiği gibi davranmasını sağlamaktadır. Bir kullanıcı bir veriyi sorguladığında veya güncellediğinde, diğer kullanıcıların aynı veriyi farklı bir durumda görmemesini garanti etmektedir.

Dağıtılmış sistemlerde tutarlılığı sağlamak zor olabilmektedir çünkü veriler farklı düğümlerde veya sunucularda saklanmakta ve bu düğümler arasında iletişim gecikmeleri veya ağ sorunları yaşanabilmektedir. Bu nedenle, tutarlılığı sağlamak için genellikle işlem protokolleri, veri replikasyonu ve senkronizasyon mekanizmaları gibi teknikler kullanılmaktadır. Bu teknikler, veri güncellemelerinin ve işlemlerin doğru sıra ve zamanlamada gerçekleştirilmesini sağlamakta, böylece sistemdeki verilerin tutarlılığı korunmaktadır (Pritchett., 2008).

3.4.3.2. Erişilebilirlik

Erişilebilirlik, dağıtılmış sistemlerde kritik bir gerekliliktir ve her istemcinin sistem üzerindeki kaynaklara her zaman erişebilmesini sağlamaktadır. Bu kavram, sistemdeki sunucu veya düğümlerin herhangi bir kesinti olmadan çalışır durumda olması gerektiğini ifade etmektedir.

Dağıtılmış sistemlerde erişilebilirlik sağlamak zor olabilmektedir çünkü sistem genellikle farklı coğrafi bölgelerde veya farklı veri merkezlerinde dağıtılmaktadır. Bu durumda, ağ kesintileri, sunucu hataları veya yazılım sorunları gibi beklenmeyen durumlar sistemde erişim sorunlarına neden olabilmektedir. Erişilebilirlik, bu tür durumlarla başa çıkabilmek için yüksek kullanılabilirlik mimarileri, yedekleme stratejileri ve otomatik iyileştirme mekanizmaları gibi teknik çözümler gerektirmektedir (Alonso ve Kuno, 2013).

3.4.3.3. Bölünebilme toleransı

Erişilebilirlik, dağıtılmış sistemlerde kritik bir gerekliliktir ve her istemcinin sistem üzerindeki kaynaklara her zaman erişebilmesini sağlamaktadır. Bu kavram, sistemdeki

sunucu veya düğümlerin herhangi bir kesinti olmadan çalışır durumda olması gerektiğini ifade etmektedir.

Bölünme Toleransı, dağıtık sistemlerin temel özelliklerinden biridir ve sistemdeki bileşenler arasındaki iletişimin kesilmesi durumunda bile sistemlerin çalışabilirliğini ve hizmet verebilirliğini sürdürebilme yeteneğini ifade etmektedir. Bu özellik, genellikle ağdaki kesintiler, düğüm hataları veya benzeri durumlar sonucunda oluşan bölünmelerden sonra dahi sistemin devam eden işlevselliğini sağlamaktadır.

Dağıtık sistemlerde bölünme toleransı sağlamak için çeşitli stratejiler ve teknikler kullanılmaktadır. Örneğin, verilerin farklı düğümler arasında bölüştürülmesi veya veri kopyalarının oluşturulması gibi yöntemler bu özelliği desteklemektedir. Bu sayede, bir düğümde meydana gelen ağ kesintisi veya hata durumunda diğer düğümler hala hizmet verebilir durumda olabilmektedir. Ayrıca, yük dengeleme algoritmaları, veri replikasyonu stratejileri ve bölge bazlı dağıtım teknikleri de bölünme toleransının sağlanmasında etkili rol oynamaktadır.

Bölünme toleransı, modern dağıtık sistemlerin dayanıklılığını ve sürekliliğini artırmaktadır. Bu özellik, sistemlerin geniş çaplı kullanıcı trafiği veya ağdaki hatalar gibi zorlayıcı koşullar altında bile hizmet vermeye devam edebilmesini sağlamaktadır. Böylece, kullanıcı deneyimi iyileşmekte ve sistemlerin güvenilirliği artmaktadır. Bu nedenle, bölünme toleransı, dağıtık sistemlerin tasarımı ve işletilmesinde kritik bir öneme sahip olup, sistemlerin dayanıklılığını sağlamak için vazgeçilmez bir gerekliliktir (Smith ve Brown, 2023).

3.5. NoSQL’de Ölçeklendirme

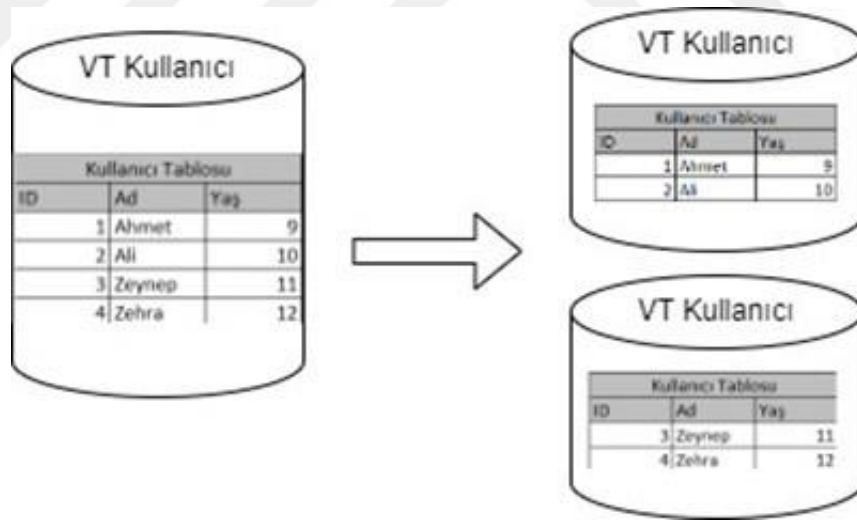
3.5.1. Parçalama

Parçalama, büyük ölçekli veritabanlarının yönetiminde kullanılan önemli bir tekniktir ve özellikle dağıtık sistemlerde performansı artırmak ve ölçeklenebilirliği sağlamak için tercih edilmektedir. Bu işlem, veritabanı tablolarını veya indekslerini belirli kriterlere göre mantıksal veya fiziksel olarak parçalara ayırma sürecini ifade etmektedir. Örneğin, aralık parçalama yöntemi, tarih alanına göre tabloyu parçalara bölerek veriyi belirli bir tarih aralığına göre yönetmeyi mümkün kılmaktadır. Bu şekilde, 2020 ve 2021 gibi farklı yıllara ait veriler farklı parçalar üzerinde saklanabilmektedir.

Hash parçalama ise veritabanındaki belirli bir alanın (örneğin, kullanıcı kimliği veya coğrafi konum) değerlerine göre veriyi bölme tekniğidir. Bu yöntemde, parçalama anahtarları belirlenmekte ve veri, bu anahtarlar kullanılarak ilgili parçaya dağıtılmaktadır. Parçalar genellikle farklı fiziksel sunucular arasında veya bulut ortamlarında dağıtılarak yük dengelemesi sağlanmaktadır.

Parçalama işlemi tamamlandıktan sonra, veritabanı yöneticileri parçaları etkin bir şekilde yönetmekte ve gerektiğinde yük dengelemesi yapmaktadır. Örneğin, bir parçanın trafik yoğunluğu artarsa, bu parça daha fazla kapasiteye sahip bir sunucuya taşınabilir veya yeni sunucular eklenerek performans artırılabilir. Bu yönetim süreci, veritabanının sürekli olarak optimize edilmesini ve genişlemesini sağlamaktadır.

Doğru parçalama anahtarlarının seçilmesi ve etkili yönetim stratejilerinin uygulanması, veritabanı performansını artırırken, aynı zamanda sistem güvenilirliğini ve ölçeklenebilirliğini de sağlamaktadır. Bu nedenle, parçalama teknolojisi, modern veri yönetimi uygulamalarında kritik bir rol oynamakta ve büyük ölçekli veri işleme ihtiyaçlarını karşılamak için yaygın olarak kullanılmaktadır (Smith ve Brown, 2022).



Şekil 3.2. Parçalama Mimarisi.

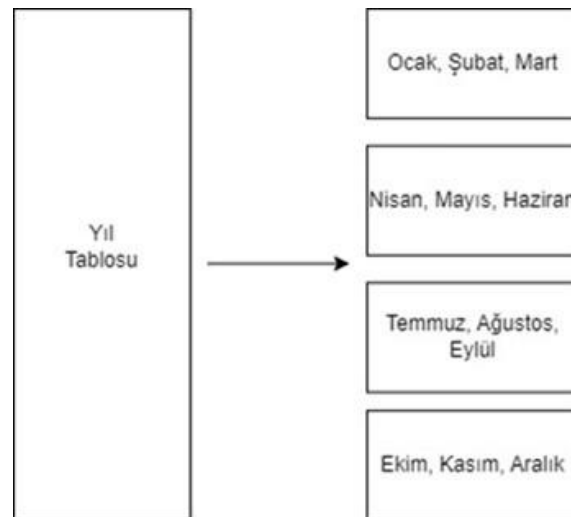
3.5.2. Bölümleme

Bölümleme, bir veritabanının verilerini belirli bir kurala göre mantıksal veya fiziksel olarak parçalara ayırma işlemidir. Bu teknik, veritabanlarının performansını artırabilir ve ölçeklenebilirliğini sağlayabilir. İlişkisel veritabanlarında, bölümleme genellikle bir tablonun daha küçük ve yönetilebilir parçalara bölünmesini içermektedir. Örneğin,

müşteri ayrıntıları tablosu temel bilgiler ve adresler gibi veriler için farklı bölümlere ayrılabilir.

NoSQL veritabanlarında, bölümleme, verilerin dağıtık sistemde her bir düğümde depolanmadığı, ancak farklı düğümler arasında bölüştürüldüğü kritik bir rol oynamaktadır. Burada, belirli kurallar ve anahtarlar kullanılarak verilerin nerede depolanacağı belirlenmektedir. Mantıksal bölümleme yöntemi, veri tablosunu veya indeksi belirli bir kriter veya anahtar değere göre parçalara ayırmaktadır. Örneğin, coğrafi konum veya zaman damgası gibi bir veri özelliğine göre mantıksal olarak parçalanabilir. Fiziksel bölümleme ise, veri tablosunu veya indeksi gerçek depolama birimlerine (örneğin, disk bölümleri veya farklı sunucular) bölerek yükü dengeleyebilmekte ve performansı artırabilmektedir.

Bölümleme, büyük ölçekli veritabanlarında sıkça kullanılan bir tekniktir çünkü veritabanı büyüdükçe veya yük arttıkça performansı artırabilir ve veri yönetimini daha etkin hale getirebilmektedir. Doğru bölümleme stratejileri seçilerek veritabanının işlemleri optimize edilmekte ve veri erişimi hızlanmaktadır. Bu yöntem aynı zamanda, veritabanı yöneticilerine veri dağıtımını ve yük dengelemesini yönetme esnekliği sağlamakta, böylece sistemlerin güvenilirliği ve kullanılabilirliği artırılmaktadır. Sonuç olarak, bölümleme teknolojisi, modern veri yönetimi uygulamalarında önemli bir yer tutmakta ve büyük veri işleme gereksinimlerini karşılamak için kritik bir rol oynamaktadır (Jones ve Smith, 2020).



Şekil 3.3. Bölümleme Mimarisi.

3.5.3. Kümeleme

Veritabanı kümeleme, bir sistemde birden fazla veritabanı örneğini veya sunucusunu bir araya getirme işlemidir. Bu yaklaşım, genellikle büyük ölçekli sistemlerde (örneğin, web veya mobil uygulamalar) performansı artırmak ve yükü dengelemek amacıyla kullanılmaktadır. Veritabanı kümeleme, genellikle "master" olarak adlandırılan merkezi bir veritabanı sunucusu tarafından yönetilen birden fazla veritabanı örneği veya sunucusunun paralel olarak çalışmasını sağlamaktadır.

Büyük sistemlerde tek bir veritabanı sunucusunun yetersiz kalabileceği durumlar için, veritabanı kümeleme çözümleri önemli bir rol oynamaktadır. Birden fazla sunucunun veya veritabanı örneğinin kullanılması, sistemdeki yükü dağıtarak performansı artırmakta ve hizmet sürekliliğini sağlamaktadır. Ancak, bu tür bir yapıyı uygulamanın ana dezavantajı, sisteme ek karmaşıklık getirmesidir. Birden fazla veritabanı sunucusunun koordinasyonu ve veri akışının yönetimi, sistem tasarımcıları için ek yönetim ve yapılandırma zorlukları doğurabilmektedir.

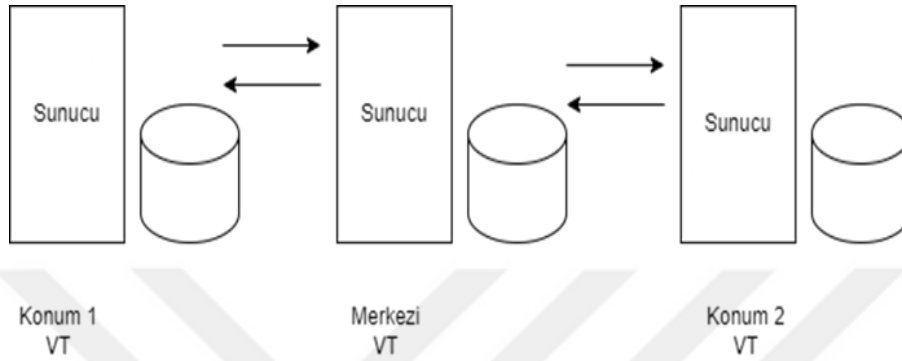
Ek karmaşıklığın üstesinden gelmek için, genellikle veritabanı kümeleme çözümleri, sistemin genelinde veri akışını izleyen ve yöneten daha üst düzey bir sunucu veya yönetim katmanı ile entegre edilmektedir. Bu yönetim katmanı, veritabanı sunucuları arasında veri senkronizasyonunu sağlamakta, yük dengelemesi yapmakta ve sistem genelinde hizmet kalitesini temin etmektedir. Böylece, veritabanı kümeleme, büyük sistemlerin performansını artırırken, aynı zamanda sistem yönetimini daha etkin ve güvenilir hale getirmektedir (Smith ve Brown, 2019).

3.5.4. Replikasyon

Veritabanı replikasyonu, bir bilgisayar veya sunucudaki veritabanının içeriğinin düzenli olarak başka bir bilgisayara veya sunucuya kopyalanması işlemidir. Bu süreç, verilerin güvenilirliğini artırmak, yüksek kullanılabilirlik sağlamak ve veri erişimini iyileştirmek amacıyla kullanılmaktadır. Replikasyon, dağıtılmış bir veritabanı ortamında tüm kullanıcıların aynı veri düzeyini paylaşmasını sağlamaktadır.

Veritabanı replikasyonunun oluşturulması ve yönetilmesi birçok unsuru içermektedir. İlk olarak, replikasyon stratejisi belirlenmelidir; bu strateji verilerin hangi sıklıkla ve hangi yöntemlerle kopyalanacağını tanımlamaktadır. Veritabanı yöneticileri, veri bütünlüğünü korumak için replikasyon işlemlerini düzenli olarak izlemeli ve yönetmelidir. Ayrıca, veri kopyalarının senkronizasyonu ve tutarlılığı sağlamak için uygun mekanizmalar kullanılmalıdır.

Veritabanı replikasyonu, dağıtılmış bir veritabanı mimarisinde önemli bir rol oynamaktadır. Çünkü kullanıcıların aynı anda ve hızlı bir şekilde güncel verilere erişimini sağlamaktadır. Bu sayede, kullanıcılar kendi görevleriyle ilgili verilere müdahale etmeden kolayca erişebilmektedirler. Ancak, replikasyon süreci veritabanı sistemleri için ek yük ve karmaşıklık da getirebilmektedir, bu nedenle doğru strateji ve yönetim önemlidir (Gray ve ark., 1996).



Şekil 3.4. Replikasyon Mimarisi.

3.6. Veritabanı Çeşitleri

3.6.1. Anahtar-Değer

Anahtar-değer veritabanları, verileri anahtar-değer çiftleri şeklinde depolayan ve erişen veritabanı sistemleridir. Her bir veri parçası, bir anahtar tarafından tanımlanmaktadır ve bu anahtar, ilgili veriye hızlı erişim sağlamaktadır. Genellikle bu tür veritabanları, bellek tabanlı veya disk tabanlı olarak uygulanabilmektedir. Bellek tabanlı uygulamalar, yüksek performanslı veri depolama ve erişim sağlarken, disk tabanlı uygulamalar daha büyük veri setleri için daha uygun olmaktadır.

Bu tür veritabanları, basit işlemler ve düşük karmaşıklık gerektiren uygulamalarda yaygın olarak kullanılmaktadır. Örneğin, önbellekleme, oturum yönetimi, ayarlar depolama ve sıralama gibi durumlarda sıklıkla kullanılabilmektedirler. Anahtar-değer veritabanlarının bir diğer avantajı, genellikle hızlı ve ölçeklenebilir olmalarıdır, bu da yüksek trafiğe ve büyük veri hacimlerine uygun olmalarını sağlamaktadır (Smith ve Brown, 2019).

3.6.2. Geniş-Sütun

Geniş-sütun veritabanı, bir tür NoSQL veritabanı modelidir ve ilişkisel veritabanı modellerine alternatif olarak geliştirilmiştir. Geniş sütun veritabanları, yüksek

ölçeklenebilirlik, esneklik ve yüksek performans gerektiren büyük veri depolama ve analiz işlemleri için idealdir.

Geniş sütun veritabanları, ilişkisel veritabanlarından farklı olarak, tabloları satır bazında değil, sütun bazında depolamaktadır. Bu, her satırın farklı sütunlara sahip olabileceği ve her sütunun birçok farklı satır içerebileceği anlamına gelmektedir. Bu yapı, belirli sorguların sütunlara daha hızlı erişim sağlamasına ve sorgu performansının artmasına olanak tanımaktadır.

Geniş sütun veritabanlarının bu esnek yapısı, büyük veri kümeleri üzerinde analiz ve sorgulama yaparken yüksek performans elde edilmesini sağlamaktadır. Bu özellikler, geniş sütun veritabanlarını büyük veri işleme, veri analitiği ve yüksek trafikli uygulamalar için ideal bir seçenek haline getirmektedir (Brown ve Johnson, 2021).

3.6.3. Doküman

Doküman veritabanı, ilişkisel veritabanlarından farklı olarak, JSON veya BSON gibi doküman formatlarını kullanarak verileri depolamaktadır. Her bir belge, bir anahtar-değer çifti olarak düşünülebilir ve belgelere ilişkin veri yapısı, her bir belgenin içinde yer almaktadır.

Doküman veritabanları, dinamik şemaya izin vermektedir; yani, her doküman kendi içinde farklı alanları veya özellikleri içerebilmektedir. Bu, uygulama gereksinimlerine uyacak şekilde veri modelinin esnek bir şekilde yapılandırılabilmesini sağlamaktadır.

Doküman veritabanları genellikle web uygulamaları, içerik yönetim sistemleri, kişisel bilgi yönetimi araçları gibi yapıları desteklemek için kullanılmaktadır. Özellikle, doküman tabanlı veritabanlarını tercih eden projeler, çeşitli ve genellikle karmaşık yapıdaki verileri depolamak ve yönetmek için uygun bir seçenek olmaktadır (Chodorow, 2013).

3.6.4. Grafik

Grafik veritabanı, graf teorisi temellerine dayanan ve ilişkisel veritabanlarından farklı bir veri modeli kullanarak veriyi depolayan bir tür NoSQL veritabanıdır. Grafik veritabanları, düğümler adı verilen varlık veya nesnelerin ve bu düğümleri birbirine bağlayan kenarlar adı verilen ilişkilerden oluşmaktadır.

Bu veritabanı türünde, verilerin yapısı ağ şeklinde ifade edilmektedir. Her düğüm, bir varlığı veya bir nesneyi temsil ederken, her kenar ise düğümler arasındaki ilişkiyi

göstermektedir. Bu sayede karmaşık ilişkiler ve ağ yapıları doğru bir şekilde modellenebilmektedir.

Grafik veritabanları, özellikle ilişkisel veritabanlarından daha verimli çalışabilmektedir çünkü ilişkiler doğrudan saklandığı ve sorgulandığı için, ilişkisel veritabanlarındaki gibi birleştirme işlemleri gerektirmemektedir. Ayrıca, grafik veritabanları genellikle çok karmaşık ve derin ilişkileri olan verileri daha etkili bir şekilde işleyebilmektedir.

Bu özellikleri nedeniyle grafik veritabanları, sosyal ağ analizleri, dolandırıcılık tespiti, öneri sistemleri ve biyoinformatik gibi alanlarda yaygın olarak kullanılmaktadır. Grafik veritabanları, veri yapılarının doğası gereği, ilişkilerin ve bağlantıların hızlı ve verimli bir şekilde sorgulanmasını ve işlenmesini sağlamaktadır (Brown ve Johnson, 2022).

3.6.5. Mekansal

Mekansal veritabanı, coğrafi verilerin depolanması, yönetilmesi ve sorgulanması için özel olarak tasarlanmış bir tür veritabanıdır. Bu tür veritabanları, coğrafi nesneleri (yerler, sınırlar, yollar, vs.) temsil eden veri türlerini desteklemekte ve bu nesneler üzerinde mekansal sorgular yapılmasını sağlamaktadır.

Mekansal veritabanları, coğrafi bilgi sistemleri (CBS) uygulamaları ve coğrafi veri analizi için kullanılmaktadır. Bu tür uygulamalar, haritalama, konum tabanlı hizmetler, arazi analizi, yol yönetimi ve çevresel modelleme gibi birçok farklı alanda yaygın olarak kullanılmaktadır.

Mekansal veritabanları genellikle coğrafi verileri depolamak için özel veri türlerine ve indeksleme tekniklerine sahiptir. Bu sayede, coğrafi nesnelerin geometrisini (nokta, çizgi, alan) ve bunların ilişkilerini (kavşak, içerme, çakışma) etkin bir şekilde işleyebilmektedir. Bu özellikler, mekansal veritabanlarının coğrafi verilerin hızlı ve doğru bir şekilde sorgulanmasını ve analiz edilmesini sağlamaktadır (Johnson ve Smith, 2023).

3.6.6. Blokzincir

Blokzincir veritabanı, birbirine bağlı bloklar halinde organize edilen ve merkezi olmayan bir ağ üzerinde dağıtılan verilerin depolanmasını sağlayan bir veritabanıdır. Bu veritabanı türü, güvenilirlik, şeffaflık ve merkezi olmayan veri yönetimi sağlamak için kullanılmaktadır.

Blokszincir, blok adı verilen veri gruplarının zaman damgası ve şifreleme yöntemleriyle birbirine bağlandığı bir zincir şeklinde çalışmaktadır. Her blok, kendisinden önce gelen bloğun veri yapısını ve benzersiz bir kimlik bilgisini içermektedir. Bu, verinin geriye dönük olarak değiştirilemez olmasını sağlamaktadır.

Blokszincir veritabanları genellikle kripto para birimleri gibi dijital varlıkların işlenmesi ve saklanması için kullanılmaktadır, ancak kullanım alanları giderek genişlemektedir. Mali işlemler, oy kullanımı, tedarik zinciri yönetimi, dijital kimlik yönetimi ve akıllı sözleşmeler gibi birçok farklı uygulama alanında blokszincir teknolojisi kullanılmaktadır.

Blokszincir veritabanlarının temel avantajları arasında veri bütünlüğü, değişmezlik, şeffaflık ve merkezi olmayan yapı bulunmaktadır. Bu özellikler, güvenilir ve izlenebilir bir veri yönetimi sistemi sağlamaktadır. Blokszincir teknolojisi, geleneksel veritabanlarına göre daha güvenli ve dayanıklı bir veri depolama yöntemi sunmaktadır (Nakamoto, 2008).

3.6.7. Bellek-İçi

Bellek içi veritabanı, verileri bellekte saklayan ve işleyen bir veritabanı türüdür. Geleneksel disk tabanlı veritabanlarının aksine, bellek içi veritabanları, disk üzerinde veri saklamak yerine, bilgisayar belleğinde veriyi tutmaktadır. Bu, veriye erişim sürelerini dramatik olarak azaltmakta ve işlem performansını artırmaktadır.

Bellek içi veritabanları, yüksek performanslı ve düşük gecikmeli işlemler gerektiren uygulamalar için ideal olmaktadır. Özellikle finansal hizmetler, telekomünikasyon, oyun endüstrisi ve gerçek zamanlı analiz gibi alanlarda kullanılmaktadır. Örneğin, finansal ticaret sistemleri hızlı işlem yapmayı gerektirirken, oyun platformları anlık tepki sürelerine ihtiyaç duymaktadır.

Bu tür veritabanları, bellek kısıtlamalarını dikkate alarak veri yönetimine öncelik vermektedir. Veriler, disk tabanlı veritabanlarından daha hızlı erişilebilir ve işlenebilir olduğundan, bellek içi veritabanları sık kullanılan verileri bellekte tutarak kısa erişim süresi sağlamaktadır. Ancak, bellek kısıtlamaları nedeniyle genellikle disk tabanlı veritabanları kadar büyük veri kümelerini işleyememektedirler. Buna karşın, yüksek bellek tüketimi maliyeti artırmaktadır. Bellek-içi veritabanları, sık erişilen ve hızlı işlenmesi gereken veriler için mükemmel bir çözüm sunmakta olup, performans

avantajları nedeniyle belirli uygulama alanlarında tercih edilmektedir (Hass ve Dayal, 2013).

3.6.8. Vektör

Vektör veritabanı, vektörel verilerin depolanması, işlenmesi ve sorgulanması için özel olarak tasarlanmış bir tür veritabanıdır. Vektör veritabanları, genellikle büyük boyutlu vektörel veri kümeleri üzerinde etkili işlemler yapmak için optimize edilmiştir.

Vektör veritabanları, özellikle makine öğrenimi, yapay zeka ve büyük veri analitiği gibi alanlarda yaygın olarak kullanılan vektörel veri yapılarının (örneğin, özellik vektörleri, gömme vektörleri, benzerlik vektörleri) depolanması ve işlenmesi için kullanılmaktadır.

Bu tür veritabanları, vektörel verilerin hızlı bir şekilde depolanması ve sorgulanması için optimize edilmiş özel algoritmalar ve veri yapıları içermektedir. Bu, büyük boyutlu vektörel veri kümeleri üzerinde etkili işlemler yapılmasını sağlamaktadır.

Vektör veritabanları, özellikle karmaşık veri yapılarına sahip ve yüksek boyutlu vektörlerin işlendiği uygulamalar için idealdir. Örnek kullanım alanları arasında görüntü işleme, metin analizi, duygu analizi, öneri sistemleri ve benzerlik arama gibi alanlar bulunmaktadır. Bu veritabanları, vektörel verilerin depolanması ve işlenmesi konusundaki ihtiyaçlara yönelik optimize edilmiş çözümler sunarak, hızlı ve etkili veri yönetimi sağlamaktadır (Johnson ve ark, 2019).

3.6.9. Nesne tabanlı

Nesne yönelimli veritabanı, nesne yönelimli programlama prensiplerine dayanan veri modelini kullanan bir tür veritabanıdır. Bu tür veritabanları, verileri nesne olarak saklamakta ve bu nesneler arasındaki ilişkileri ve davranışları modellemektedir.

Nesne yönelimli veritabanları, geleneksel ilişkisel veritabanlarından farklıdır. İlişkisel veritabanlarında veri, tablolar ve bu tablolardaki satır ve sütunlar kullanılarak modellenirken, nesne yönelimli veritabanlarında gerçek dünyadaki nesneler ve bunların birbirleriyle olan ilişkileri doğrudan modelleme yapısı olarak kullanılmaktadır.

Bu tür veritabanları, nesne yönelimli programlama dilleriyle (örneğin, Java, C++, Python) entegrasyonu kolaylaştırmaktadır. Veritabanındaki nesneler, programlama dilindeki nesnelerle doğrudan ilişkilendirilebilmekte ve veriye erişim için nesne yönelimli sorgu dilleri kullanılabilmektedir.

Nesne yönelimli veritabanlarının kullanım alanları arasında karmaşık veri yapıları, nesne tabanlı uygulamalar, modelleme ve simülasyon gibi alanlar bulunmaktadır. Özellikle büyük ve karmaşık veri yapılarını saklamak, yönetmek ve sorgulamak için uygun olan bu veritabanları, nesne yönelimli programlama prensiplerine dayalı sistemlerin gereksinimlerini karşılamak için idealdir (Paterson ve ark., 2009).

3.6.10. Zaman-Serisi

Zaman serisi veritabanı, zamana bağlı veri noktalarını saklayan, işleyen ve sorgulayan bir tür veritabanıdır. Zaman serisi veritabanları, zamanla değişen veri noktalarını kronolojik olarak saklamak ve bu verilere erişmek için optimize edilmiştir.

Zaman serisi veritabanları, genellikle büyük miktarda zaman serisi verisinin etkin bir şekilde saklanması, analiz edilmesi ve sorgulanması gereken uygulamalarda kullanılmaktadır. Örnek olarak, finansal piyasaların izlenmesi, enerji tüketimi analizi, ağ performans izleme ve sağlık izleme gibi alanlar bu tür veritabanlarını gerektirebilir.

Zaman serisi veritabanları, zaman serisi verilerini depolamak ve bu verilere hızlı bir şekilde erişmek için özel olarak tasarlanmış veri yapıları ve indeksleme teknikleri kullanılmaktadır. Zaman serisi veritabanları, veri sıkıştırma, örnekleme ve veri segmentasyonu gibi özel teknikleri kullanarak büyük veri kümelerini etkili bir şekilde yönetmektedir. Ayrıca, zaman serisi veritabanları genellikle paralel ve dağıtık hesaplama yeteneklerine sahiptir, bu da büyük ölçekli veri analizi gereksinimlerini karşılamaktadır.

Bu veritabanları, zamanla değişen verilerin hızlı ve verimli bir şekilde işlenmesini ve analiz edilmesini sağlayarak, zamana dayalı veri analizleri için ideal bir çözüm sunmaktadır (Shieh ve Larson, 2017).

3.6.11. NewSQL

NewSQL, geleneksel SQL veritabanları ile NoSQL veritabanları arasında bir orta noktada bulunan bir veritabanı türüdür. NewSQL veritabanları, yüksek performans, ölçeklenebilirlik ve ACID uyumluluğu gibi özellikleri bir araya getirerek, büyük ölçekli işlemler için güvenilir bir çözüm sunmayı amaçlamaktadır.

NewSQL terimi, büyük ölçekli web hizmetleri ve bulut tabanlı uygulamalar gibi modern uygulamaların gereksinimlerini karşılamak için geleneksel SQL veritabanlarını geliştirmek üzere kullanılmaktadır. Bu tür veritabanları, geleneksel SQL veritabanlarının performans kısıtlamalarını aşmak için ölçeklenebilir mimarilere,

paralel sorgulama yeteneklerine ve dağıtılmış veritabanı mimarilerine odaklanmaktadır.

NewSQL veritabanları, yüksek paralel işlem yetenekleri, dağıtılmış veri saklama ve işleme, otomatik ölçeklendirme gibi özellikler sunmaktadır. Bu, büyük miktarda veriye hızlı erişim sağlamak ve yüksek talepleri karşılamak için önemlidir. Ayrıca, ACID uyumluluğu sayesinde veri bütünlüğünü korumakta ve güvenilir işlemler sağlamaktadır.

NewSQL veritabanları, geleneksel SQL veritabanlarından farklı olarak, daha modern ve esnek mimarilere sahiptir. Bu, veritabanının performansını artırırken aynı zamanda işletme maliyetlerini düşürmektedir. NewSQL veritabanları, modern uygulamaların yüksek performans ve ölçeklenebilirlik gereksinimlerini karşılamak için ideal bir çözüm sunmaktadır (Stonebraker ve Çetintemel, 2005).

3.6.12. SQL

SQL veritabanı, verilerin ilişkisel veri tabanı yönetim sistemi (RDBMS) kullanılarak depolandığı ve yönetildiği bir tür veritabanıdır. SQL, veritabanı işlemlerini gerçekleştirmek için standartlaştırılmış bir sorgulama dilidir ve ilişkisel veritabanlarında veri manipülasyonunu, sorgulamayı ve yönetimi sağlamaktadır.

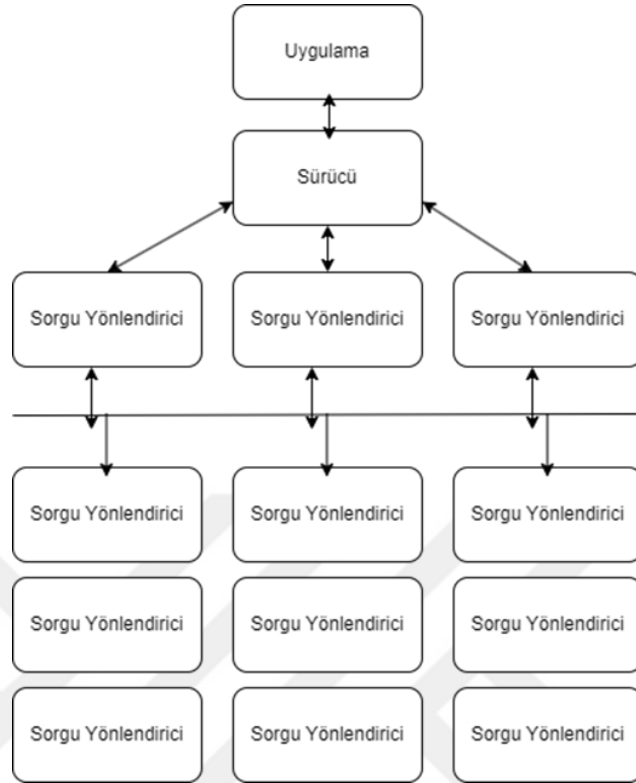
SQL veritabanları, tablolar, satırlar ve sütunlar şeklinde yapılandırılmış verileri kullanmaktadır. Veritabanı yöneticileri ve kullanıcılar, SQL sorguları aracılığıyla veritabanıyla etkileşimde bulunabilmekte ve veri manipülasyonu yapabilmektedirler. SQL, veri ekleme, veri güncelleme, veri silme ve veri sorgulama gibi temel veritabanı işlemlerini gerçekleştirmek için kullanılmaktadır.

SQL veritabanları, ACID (Atomicity, Consistency, Isolation, Durability) özelliklerini desteklemektedir, yani işlem bütünlüğünü, tutarlılığını, izolasyonunu ve kalıcılığını sağlamaktadır. Bu özellikler, veri bütünlüğünü korumakta ve veritabanı işlemlerinin güvenilirliğini artırmaktadır.

SQL veritabanları, geniş bir uygulama yelpazesinde kullanılmakta olup, özellikle finansal sistemler, e-ticaret platformları, müşteri ilişkileri yönetimi (CRM) sistemleri ve daha birçok alanda yaygın olarak tercih edilmektedir. Bu veritabanları, karmaşık veri yapıları ve ilişkileri yönetmek için ideal bir çözüm sunmakta ve verinin etkin bir şekilde depolanmasını, sorgulanmasını ve yönetilmesini sağlamaktadır (Silberschatz ve ark., 2020).

3.7. MongoDB

3.7.1. Mimari



Şekil 3.5. MongoDB Mimarisi.

MongoDB, geleneksel tablo tabanlı yapılar yerine belge odaklı bir veri modeli kullanarak verilerin esnek ve JSON benzeri belgelerde saklanmasını sağlamaktadır. Bu esneklik, dinamik veri gereksinimlerine uyum sağlama ve kullanım kolaylığı açısından önemli avantajlar sunmaktadır. Verilerin yatay olarak bölünmesi için parçalama tekniğini kullanan MongoDB, büyük veri kümelerinin yönetimini kolaylaştırarak veri miktarı arttıkça sistemin ölçeklenmesini mümkün kılmaktadır.

Yerleşik replikasyon ve otomatik yük devretme özellikleri ile bir düğüm arızası durumunda bile verilerin erişilebilirliği korunmakta, böylece kesintisiz veri erişimi ve operasyonel süreklilik sağlanmaktadır. Primary-secondary protokolü sayesinde yoğun okuma işlemlerinde replikalar kullanılabilmekte, bu da okuma yükünü dağıtarak veritabanı performansını artırmaktadır. MongoDB, replika kümeleri aracılığıyla güçlü tutarlılık sağlamakta ve tüm replika kümesi üyelerinin aynı verilere sahip olmasını garanti etmektedir.

Belge düzeyinde atomik yazma işlemleri sunan MongoDB, eşzamanlı yazma işlemlerinin atomik ve birbirinden yalıtılmış olmasını garanti ederek veri tutarlılığı ve

güvenilirliğini artırmaktadır. Esnek ve dinamik veri modellemesi, sistemin genelini etkilemeden şemada değişiklik yapılmasına izin vermekte, çok çeşitli indeksleme seçenekleri ise veri erişimini hızlandırmaktadır. Yerleşik toplama çerçevesi ile veri analizi ve raporlama için güçlü araçlar sunmaktadır.

MongoDB, modern uygulamaların gereksinimlerini karşılamak üzere tasarlanmış olup, yüksek performans, esneklik ve ölçeklenebilirlik özellikleri ile öne çıkmaktadır (Chodorow, 2013).

3.7.2. Veri modeli

MongoDB, belgelere dayalı bir veri modeli kullanmaktadır. Veriler BSON formatında depolanmaktadır. BSON, JSON'un binarize edilmiş halidir ve MongoDB'nin performansını artırmak için kullanılmaktadır. Her bir belge, bir veya daha fazla alan içerebilir. Alanlar, çeşitli veri tiplerini desteklemektedir; bunlar arasında stringler, sayılar, tarihler, diziler, ikili veriler ve hatta başka belgeler (alt belgeler) yer almaktadır.

Benzer bir yapıyı paylaşma eğiliminde olan belgeler koleksiyonlar olarak düzenlenmektedir. Koleksiyonları, ilişkisel bir veritabanındaki bir tabloya benzer olarak düşünmek faydalı olabilir: belgeler satırlara ve alanlar sütunlara benzemektedir. MongoDB'deki bu esnek yapı, dinamik veri gereksinimlerine uyum sağlama ve kullanım kolaylığı açısından önemli avantajlar sunmaktadır (McClarence, 2023).

3.7.3. Sorgu modeli

MongoDB, geliştirmeyi doğal hale getirmek için tüm popüler programlama dilleri ve çerçeveleri destekleyen yerel sürücüler sunmaktadır. Bu sürücüler arasında Java, NET, Ruby, PHP, JavaScript, Node.js, Python, Perl, Scala ve diğerleri bulunmaktadır. Ayrıca, 30'dan fazla topluluk tarafından geliştirilmiş sürücü de mevcuttur. MongoDB sürücülerini, ilgili programlama dili için deyimisel olacak şekilde tasarlanmıştır. İlişkisel veritabanlarıyla karşılaştırıldığında, MongoDB sorgu modelinin temel bir farkı, sorguların veritabanının API'si içinde yöntemler veya işlevler olarak uygulanmasıdır. Bu, SQL gibi tamamen ayrı bir dil yerine belirli bir programlama dili kullanılarak yapılmaktadır. MongoDB'nin JSON belge modeli ile nesne yönelimli programlamada kullanılan veri yapıları arasındaki yakınlık, uygulamalarla entegrasyonu kolaylaştırmaktadır.

Bu özellikler, MongoDB'nin modern yazılım geliştirme süreçlerine daha doğal ve uyumlu bir şekilde entegre olmasını sağlamaktadır. MongoDB'nin esnek veri modeli, güçlü sorgulama ve indeksleme yetenekleri ile birlikte, geliştiricilere veritabanı işlemlerini kolay ve etkili bir şekilde gerçekleştirme imkanı sunmaktadır (Chodorow, 2019).

3.7.4. Çoğaltma

MongoDB replikasyonu, yüksek kullanılabilirlik ve veri güvenliği sağlamak amacıyla kullanılan kritik bir özelliktir. Bu süreç, birincil bir veri kümesinin verilerinin ve işlemlerinin bir veya daha fazla ikincil veri kümesine otomatik olarak kopyalanmasını içermektedir. Bu kopyalama işlemi, herhangi bir veri kümesinin başarısız olması durumunda diğer kopyaların devreye girmesini mümkün kılmaktadır.

MongoDB replikasyonu, genellikle birincil-ikincil modeli olarak adlandırılan bir mimariyi kullanmaktadır. Bu modelde, birincil veri kümesi yazma işlemlerini gerçekleştirirken, ikincil veri kümeleri birincil veri kümesinin verilerini kopyalamakta ve okuma işlemlerini üstlenmektedir. Birincil veri kümesi başarısız olduğunda veya erişilemez olduğunda, ikincil veri kümelerinden biri otomatik olarak birincil rolünü devralarak (otomatik yenidenseçim) sistemin kesintisiz çalışmasını sağlamaktadır.

Replikasyonun sağladığı diğer önemli avantajlar arasında veri yedekleme, veri kurtarma ve yük dengeleme bulunmaktadır. Okuma işlemlerinin ikincil veri kümelerine dağıtılması, uygulama performansını artırarak sistem üzerindeki yükü dengelemektedir. Bu yapı, veri kaybı riskini minimize ederken, yüksek erişilebilirlik ve güvenilirlik sağlamaktadır. MongoDB'nin replikasyon mekanizması, modern veri tabanı yönetim sistemlerinde kritik öneme sahip olan bu gereksinimleri etkin bir şekilde karşılamaktadır.

MongoDB replikasyonu, veri yedekleme ve kurtarma süreçlerini basitleştirirken, veri güvenliği ve erişilebilirlik açısından da yüksek performans sağlamaktadır. Bu özellikler, MongoDB'nin büyük ölçekli, kritik uygulamalarda güvenilir bir çözüm olmasını sağlamaktadır (Chodorow, 2019).

3.7.5. Bölümleme

MongoDB'de bölümleme teknolojisi, büyük ölçekli veri depolama ve yüksek performanslı veri işleme gereksinimlerini karşılamak için kullanılan dağıtılmış bir veritabanı yaklaşımıdır. Bölümleme teknolojisi, bir MongoDB kümesini fiziksel

olarak birden fazla makine veya sunucuya bölerek veri ve yükün paralel olarak dağıtılmasını sağlamaktadır. Bu yöntem, veri tabanının büyüklüğü ve yükü arttıkça ölçeklenebilirliği ve performansı artırmaktadır. MongoDB bölümlene teknolojisi, temel olarak üç ana bileşenden oluşmaktadır: Bölme, "mongos" (MongoDB yönlendiricisi) ve yapılandırma sunucusu.

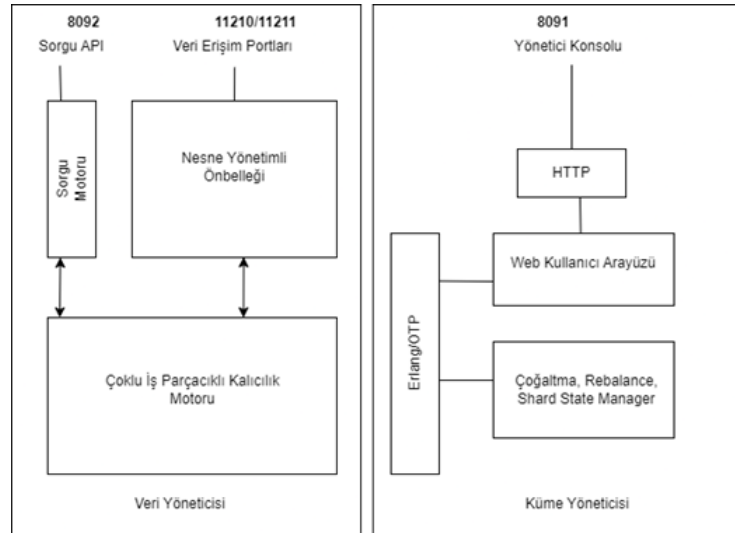
Bölme, verinin fiziksel olarak bölündüğü ve depolandığı MongoDB veri kümesinin parçalarını temsil etmektedir. Genellikle bir veya daha fazla sunucu veya makine kümesinden oluşan bölmelere, kendi veri yüklerini yönetmektedirler. Mongos, uygulamaların MongoDB bölümlene kümesiyle iletişim kurmasını sağlayan bir yönlendirici işlevi görmektedir. Uygulamalar, mongos aracılığıyla istekte bulunur ve mongos bu istekleri doğru bölmelere yönlendirir, böylece bölümlene kümesinin karmaşıklığını uygulamalardan gizlemektedir. Yapılandırma sunucusu, bölümlene kümesinin yapılandırma verilerini ve yönlendirmelerini saklamaktadır. Her bölümlene kümesi için bir veya daha fazla yapılandırma sunucusu bulunmakta ve bu sunucular, mongos'un veri dağıtımını yönetmesine ve istekleri doğru bölmelere yönlendirmesine yardımcı olmaktadır.

Bu bileşenlerin etkileşimi, MongoDB'nin büyük veri kümelerini etkili bir şekilde yönetmesini ve yüksek performanslı veri işlemleri gerçekleştirmesini mümkün kılmaktadır. Bölümlene teknolojisi, verilerin yatay olarak bölünmesini sağlamakta ve böylece büyük veri kümeleri üzerindeki yükü dağıtarak ölçeklenebilirliği artırmaktadır. Bu yapı, MongoDB'nin büyük ölçekli uygulamalarda verimli ve performanslı bir şekilde çalışmasını sağlamaktadır (Harrison & Buckingham, 2016).

3.8. Couchbase

3.8.1. Mimari

Couchbase kümeleri, birden fazla düğümden oluşmaktadır. Bir küme, mantıksal olarak yapılandırılmış bir veya daha fazla Couchbase sunucusu örneğinden oluşmaktadır. Aşağıda Couchbase sunucu mimarisi diyagramı verilmiştir:



Şekil 3.6. Couchbase Mimarisi.

Couchbase, çoğu küme teknolojisinin aksine, eşler arası bir düğüm mekanizması kullanmaktadır. Bu, kümedeki düğümler arasında herhangi bir fark olmadığı ve her düğümün aynı işlevselliği sağladığı anlamına gelmektedir. Bu yapı, tek bir hata noktasının olmamasını sağlamaktadır. Bir düğüm arızalandığında, başka bir düğüm onun sorumluluğunu üstlenmekte ve bu da yüksek kullanılabilirlik sağlamaktadır.

Eşler arası düğüm yapısı, Couchbase kümelerinin daha esnek ve dayanıklı olmasına katkıda bulunmaktadır. Bu yapı sayesinde, yük dengeleme ve veri replikasyonu işlemleri daha etkin bir şekilde yönetilebilmekte ve küme performansı optimize edilebilmektedir. Couchbase'in bu mimarisi, veri kaybı riskini minimize etmekte ve uygulamaların kesintisiz çalışmasını sağlamaktadır (Brown & Hewlett, 2017).

Veri Yöneticisi: Couchbase veritabanı sisteminde gerçekleştirilen herhangi bir işlem (okuma, ekleme, güncelleme vb.) bellek içinde saklanmakta ve bu da bir önbellek katmanı olarak işlev görmektedir. Varsayılan olarak, her belge, bellek dolana kadar bellek içinde saklanmaktadır. Bu yapı, Memcache için bir yer değiştirme çözümü sunmaktadır.

Ancak, kayıtların kalıcılığını sağlamak için disk kuyruğu adı verilen bir kavram vardır. Bu kavram, kayıtları eşzamansız olarak diske yazarak istemci taleplerini etkilemeden gerçekleştirir. Bu işlevsellik, veri yöneticisi tarafından otomatik olarak sağlanmakta ve insan müdahalesi gerektirmemektedir.

Disk kuyruğu sayesinde, Couchbase veritabanı yüksek performanslı bellek içi işlemler ile veri kalıcılığı arasında denge kurmaktadır. Bellek dolduğunda veya belirli koşullar

gerçekleştğinde veriler diske yazılarak uzun süreli saklama sağlanmaktadır. Bu yapı, veritabanının hem hızlı hem de güvenilir olmasını sağlamaktadır (Brown & Hewlett, 2017).

Küme Yöneticisi: Küme yöneticisi, bir kümedeki düğümlerin yönetiminden ve izlenmesinden sorumludur. Her Couchbase düğümü, küme yöneticisi bileşenini, veri depolama bileşenini ve veri yöneticisini içermektedir. Küme yöneticisi, veri depolama ve alımını yönetmektedir. Bellek önbellek katmanını, disk kalıcılık mekanizmasını ve sorgu motorunu içermektedir.

Couchbase istemcileri, gerekli verilerin hangi düğümde bulunduğunu öğrenmek için küme yöneticisi tarafından sağlanan küme haritasını kullanmakta ve ardından veri yöneticisi ile iletişim kurarak veritabanı işlemlerini gerçekleştirmektedir.

Küme yöneticisi, veritabanı işlemlerinin düzgün ve verimli bir şekilde yürütülmesini sağlamaktadır. Veri depolama, önbellekleme ve kalıcılık gibi kritik işlevler küme yöneticisi tarafından yönetilmektedir. Ayrıca, istemcilerin doğru düğümlere yönlendirilmesini sağlayarak veri erişimini optimize etmektedir. Bu yapı, Couchbase'in yüksek performanslı ve ölçeklenebilir bir veritabanı çözümü sunmasını mümkün kılmaktadır (Brown & Hewlett, 2017).

3.8.2. Veri modeli

Couchbase, belge odaklı bir NoSQL veritabanı olarak tasarlanmış olup, yüksek performans, esneklik ve ölçeklenebilirlik gerektiren uygulamalar için ideal bir çözüm sunmaktadır. Couchbase'in veri modeli, JSON formatındaki belgeler üzerine kuruludur ve bu belgeler üzerinde SQL benzeri sorgular yapılmasını sağlayan güçlü bir sorgu dili olan N1QL'i desteklemektedir. Bu veri modeli, aşağıdaki temel bileşenlerden oluşmaktadır:

Couchbase, birden fazla düğümden oluşan bir küme mimarisi kullanmaktadır. Veriler, bu küme içinde dağıtılarak depolanmakta ve bu sayede sistem ölçeklenebilirlik ve yüksek erişilebilirlik sağlamaktadır.

Couchbase, verileri mantıksal olarak ayrılmış bölmeler olan "Kova"larda saklamaktadır. Bu platform, Kalıcı Kovalar aracılığıyla kalıcı veri depolama sağlarken, Geçici Kovalar geçici veri depolama için kullanılmaktadır. Her bir kova, belirli bir türde veri depolama ve yönetim işlevselliği sunmaktadır.

Temel veri birimi, belge olarak adlandırılmaktadır. Her belge, benzersiz bir belge anahtarı ile tanımlanmakta ve JSON formatında saklanmaktadır. Bu yapı, esnek bir şema sağlamak ve veriye hızlı erişim imkanı sunmaktadır.

Couchbase aynı zamanda bir anahtar-değer veritabanı olarak da işlev görmektedir. Her belge, benzersiz bir anahtar ile saklanmakta ve bu anahtar kullanılarak hızlı erişim sağlanmaktadır.

N1QL, Couchbase'in sunduğu SQL benzeri sorgu dili olarak öne çıkmaktadır. N1QL ile JSON belgeleri üzerinde karmaşık sorgular, filtrelemeler ve veritabanı işlemleri gerçekleştirilebilmektedir. Bu dil, klasik SQL DML işlemlerini desteklemekte ve veri manipülasyonunu kolaylaştırmaktadır.

Couchbase ayrıca Harita-İndirgeme tabanlı görünüm ve indeksler oluşturmayı desteklemektedir. Görünüm, özel sorgular ve raporlar oluşturmak için kullanılırken, indeksler veritabanı sorgularının performansını artırmak amacıyla kullanılmaktadır.

Bu bileşenler, Couchbase'in kompleks uygulama ihtiyaçlarını karşılamak üzere tasarlanmış kapsamlı bir veri modeli sunmaktadır. Sistem, yüksek performans, esneklik ve güvenilirlik sunarak modern uygulamalar için güçlü bir altyapı sağlamaktadır (Brown & Hewlett, 2017).

3.8.3. Hata toleransı

Couchbase, yüksek erişilebilirlik ve veri bütünlüğünü sağlamak için gelişmiş hata toleransı mekanizmaları sunmaktadır. Bu mekanizmalar, veri kaybını önlemek ve hizmet sürekliliğini sağlamak amacıyla tasarlanmıştır. Couchbase'in hata toleransı sağlama stratejileri şunlardır:

Veri Çoğaltma: Couchbase, veri çoğaltma mekanizması ile hata toleransı sağlamaktadır. Her bir kova için belirlenen çoğaltma faktörü sayesinde veriler, kümedeki birden fazla düğüm üzerinde kopyalanmaktadır. Örneğin, çoğaltma faktörü 2 olarak ayarlandığında her veri parçası, iki farklı düğümde depolanmaktadır (Membrez, 2018).

Aktif ve Replika Dökümanlar: Her veri ögesi için bir asıl kopya ve bir veya daha fazla yedek kopya mevcuttur. Aktif dökümanlar, ana düğümde saklanmakta ve veri yazma işlemleri burada gerçekleştirilmektedir. Replika dökümanlar ise diğer düğümlerde saklanmakta ve aktif dökümanlar ile senkronize edilerek veri kaybı önlenmektedir (Harrison, 2014).

Otomatik Yükdevretme: Couchbase, bir düğüm arızalandığında otomatik yükdevretme işlemi gerçekleştirmektedir. Bu işlem, arızalanan düğümdeki verilerin replika kopyalarını aktif hale getirerek hizmetin kesintisiz devam etmesini sağlamaktadır. Otomatik yükdevretme, sistemdeki hata durumlarını otomatik olarak algılamakta ve müdahale etmektedir (Strauch, 2011).

Veri Kalıcılığı: Couchbase, yazma işlemlerinin kalıcılığını garanti altına almak için senkron ve asenkron kalıcılık seçenekleri sunmaktadır. Senkron kalıcılıkta, yazma işlemi verinin hem aktif hem de replika düğümlere yazılmasını beklerken, asenkron kalıcılıkta veri sadece aktif düğüme yazılmakta ve replika düğümlere asenkron olarak kopyalanmaktadır. Bu seçenekler, kullanıcıların performans ile veri dayanıklılığı arasında denge kurmalarını sağlamaktadır (Enriquez, 2015).

Veri Yeniden Dengeleme: Couchbase, küme yapılandırması değiştiğinde (örneğin, düğüm eklenmesi veya kaldırılması durumunda) veri yeniden dengeleme işlemi gerçekleştirmektedir. Bu işlem, verilerin küme boyunca eşit şekilde dağıtılmasını sağlamakta ve yük dengeleme işlevi görmektedir. Veri yeniden dengeleme, hem performans optimizasyonu hem de hata toleransı açısından önemli olmaktadır (Sadalage & Fowler, 2012).

İzleme ve Uyarılar: Couchbase, sistem yöneticilerinin küme durumunu sürekli olarak izlemelerini sağlayan kapsamlı izleme ve uyarı mekanizmaları sunmaktadır. Yönetim konsolu ve diğer izleme araçları aracılığıyla düğüm durumu, bellek kullanımı, veri çoğaltma durumu gibi önemli metrikler sürekli olarak takip edilebilmekte ve olası sorunlar hızlıca tespit edilip çözüme kavuşturulmaktadır (Dabholkar & Joshi, 2017).

Bu özellikler, Couchbase'in hata toleransı sağlama yeteneklerini güçlendirerek veri tabanı sisteminin güvenilirliğini artırmakta ve kullanıcıların kritik uygulamaları için güvenilir bir altyapı sunmaktadır (Brown & Hewlett, 2017).

3.9. Cassandra

3.9.1. Mimari

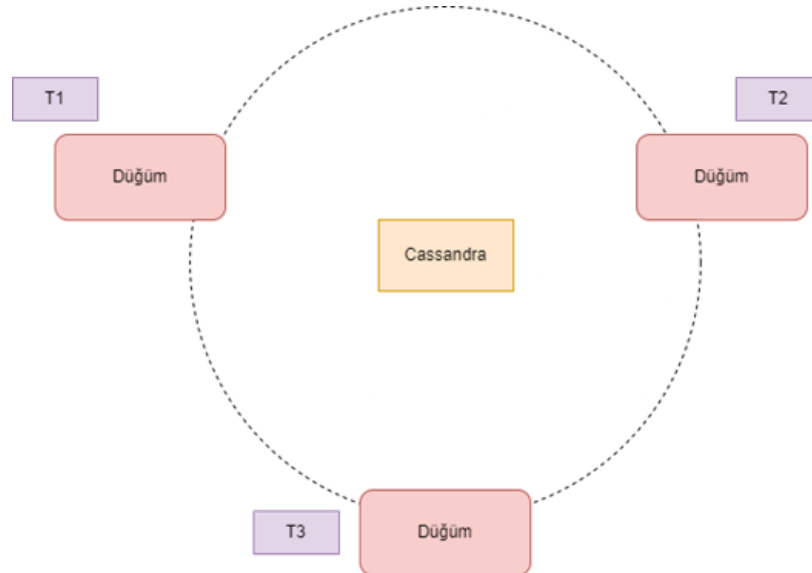
Cassandra, dağıtılmış veri depolama sistemlerinde önemli bir yer tutan bir mimariye sahip olmaktadır. Temel özellikleri arasında herhangi bir ana veya yedek düğümü olmayacak şekilde tasarlanması, halka tipi bir mimariye sahip olması ve verilerin

otomatik olarak tüm düğümlere dağıtılması bulunmaktadır. Bu yapı, veriye yüksek erişilebilirlik ve ölçeklenebilirlik sağlamaktadır.

Cassandra, verilerin dağıtımını anahtarların hash değerleri üzerinden gerçekleştirmektedir. Her anahtar, sayısal bir hash değerine eşlenmekte ve bu değer kullanılarak veri düğümler arasında dağıtılmaktadır. Bu dağıtım yöntemi, veri erişimini hızlandırırken aynı zamanda yükü dengelemekte ve veri bütünlüğünü korumaktadır.

Veriler, düğümler arasında yedeklilik sağlamak amacıyla çoğaltılmaktadır. Cassandra'nın HDFS benzeri bu özelliği, veri kaybını önlemekte ve hizmet sürekliliğini sağlamaktadır. Ayrıca, veriler bellekte geçici olarak tutulmakta ve diske yazma işlemleri İpucu Aktarımı yöntemi ile gerçekleştirilmektedir. Bu yöntem, geçici ağ sorunları veya düğüm arızaları gibi durumlarda veri kaybını önlemeye yardımcı olmaktadır.

Son olarak, Cassandra'nın halka mimarisi, her düğüme bir belirteç değeri atayarak veri dağıtımını optimize etmektedir. Bu belirteçler, düğümlerin halka şeklinde mantıksal olarak dağılmasını sağlamakta ve sistemdeki veri dağıtımını ve erişimini düzenlemektedir. Cassandra'nın bu özellikleri, büyük ölçekli dağıtılmış sistemlerde güvenilirlik, performans ve ölçeklenebilirlik sağlamak için tasarlanmaktadır.



Şekil 3.7. Cassandra Mimarisi.

Cassandra'nın mimarisi, dağıtılmış sistemlerde esneklik ve yüksek dayanıklılık sağlamak için bir dizi ek özellik sunmaktadır. Bu özelliklerden bazıları şunlardır:

Cassandra, birden fazla veri merkezini destekleyen bir yapıya sahiptir. Bu özellik, farklı coğrafi bölgelerde veya farklı ağ segmentlerinde bulunan veri merkezleri arasında veri depolamayı ve yönetmeyi mümkün kılmaktadır. Çeşitli veri merkezleri arasında veri çoğaltılabilmekte, bu da sistemde yüksek erişilebilirlik ve felaket durumlarına karşı direnç sağlamaktadır.

Veriler, Cassandra'da veri merkezleri arasında çoğaltılmaktadır. Bu, her veri parçasının birden fazla kopyasının farklı veri merkezlerinde saklanabileceği anlamına gelmektedir. Örneğin, üç kopyası bir veri merkezinde saklanırken, dördüncü kopyası uzaktan yedekleme amacıyla uzak bir veri merkezinde saklanabilmektedir. Bu düzenleme, veri merkezi seviyesindeki felaketlere karşı koruma sağlamakta ve sistem genelinde veri bütünlüğünü artırmaktadır.

Cassandra'da yapılan veri okuma işlemleri, genellikle yerel bir veri merkezinden gerçekleştirilmektedir. Bu, veriye erişim hızını artırmakta ve ağ üzerindeki trafiği azaltmaktadır. Ancak, yerel veri merkezindeki veriye erişim mümkün olmadığında veya veri kullanılabilir olmadığında, uzak bir veri merkezinden veri okuma işlemleri gerçekleştirilebilmektedir. Bu yaklaşım, sistem performansını optimize ederken aynı zamanda yüksek kullanılabilirlik ve hizmet sürekliliği sağlamayı hedeflemektedir.

Bu özellikler, Cassandra'nın mimarisinin esneklik, yüksek erişilebilirlik ve veri bütünlüğü sağlamadaki yetkinliğini göstermektedir. Dağıtılmış veri depolama sistemlerinde bu tür gelişmiş özellikler, büyük ölçekli uygulamalar ve kritik veri yönetimi senaryoları için ideal çözümler sunmaktadır (Malik, 2015).

3.9.2. Sıkıştırma

Sıkıştırma, Cassandra veri tabanındaki veri dosyalarını sıkıştıran ve optimize eden kritik bir süreçtir. Bu süreç, veri tabanında biriken silebilir veya güncellenebilir veri sürümlerini bir araya getirerek ve gereksiz veri sürümlerini temizleyerek depolama alanını etkin bir şekilde kullanmaktadır. Cassandra, dağıtılmış doğası gereği birçok düğümden oluşan bir sistemdir ve bu nedenle aynı veri parçası farklı düğümlerde farklı sürümlerde depolanmaktadır. Sıkıştırma işlemi, bu durumda ortaya çıkan veri parçalanmasını çözerek ve gereksiz veri kopyalarını temizleyerek veri tabanının bütünlüğünü ve performansını artırmaktadır.

Sıkıştırma genellikle düzenli aralıklarla veya belirli bir veri boyutu veya dosya sayısı eşiğine ulaşıldığında tetiklenmektedir. Bu süreç sırasında, Cassandra veri dosyaları

okunmakta, gereksiz veriler temizlenmekte, sıkıştırılmakta ve yeni birleştirilmiş veri dosyaları oluşturulmaktadır. Sıkıştırma, veri tabanının okuma ve yazma performansını artırırken, depolama alanının daha verimli kullanılmasını sağlamaktadır. Bu nedenle, Cassandra'da sıkıştırma sürecinin düzenli olarak yönetilmesi, sistem performansı ve veri yönetimi açısından kritik öneme sahiptir.

Bu özellikler, Cassandra'nın büyük ölçekli dağıtılmış veri tabanlarında verimliliği ve performansı artırmada nasıl önemli bir rol oynadığını göstermektedir. Sıkıştırma süreci, hem veri bütünlüğünü korumakta hem de depolama alanının etkin kullanımını sağlamaktadır. Dolayısıyla, Cassandra'nın sıkıştırma mekanizmalarının etkin bir şekilde yönetilmesi, sistemin genel sağlığı ve performansı açısından vazgeçilmezdir (Burbank, 2024).

3.9.3. Veri modeli

Sıkıştırma, Cassandra veri tabanındaki veri dosyalarını sıkıştıran ve optimize eden kritik bir süreçtir. Bu süreç, veri tabanında biriken silebilir veya güncellenebilir veri sürümlerini bir araya getirerek ve gereksiz veri sürümlerini temizleyerek depolama alanını etkin bir şekilde kullanmaktadır. Cassandra, dağıtılmış doğası gereği birçok düğümden oluşan bir sistemdir ve bu nedenle aynı veri parçası farklı düğümlerde farklı sürümlerde depolanmaktadır. Sıkıştırma işlemi, bu durumda ortaya çıkan veri parçalanmasını çözerek ve gereksiz veri kopyalarını temizleyerek veri tabanının bütünlüğünü ve performansını artırmaktadır.

Cassandra veri modelleme süreci, bir iş sürecini desteklemek için gereken veri gereksinimlerini ve erişim modellerini tanımlamak ve analiz etmek amacıyla kullanılmaktadır. Bu süreç, verilerin doğru bir şekilde yakalanmasını, depolanmasını ve işlenmesini sağlamak için detaylı bir inceleme ve planlama gerektirmektedir.

Veri modeli, bir sorunun tanımlanmasına yardımcı olarak farklı veri modelleme yaklaşımlarını değerlendirmeyi ve en uygun olanını seçmeyi sağlamaktadır. Aynı zamanda, iş süreci için gerekli olan tüm verilerin verimli bir şekilde yakalanmasını ve depolanmasını temin etmektedir. Modeller, veri tabanı şemasını oluştururken önemli kavramları ve terminolojiyi belgeleyerek uzun vadeli bakım ve geliştirme için bir temel sağlamaktadır.

Cassandra, ilişkisel olmayan bir NoSQL veritabanı olduğundan, ilişkisel veritabanları gibi tasarlanmamalıdır. Veri modelleme sürecinde, veriler gerektiğinde denormalize

edilmektedir. Bu yaklaşım, Cassandra'da yazma işlemlerinin ilişkisel veritabanlarına göre daha ekonomik olmasını sağlamakta ve böylece performansı artırmak için verilerin optimize edilmesini mümkün kılmaktadır (Robinson & Hewitt, 2016).

3.9.4. Hata toleransı

Sıkıştırma, Cassandra veri tabanındaki veri dosyalarını sıkıştıran ve optimize eden kritik bir süreçtir. Bu süreç, veri tabanında biriken silebilir veya güncellenebilir veri sürümlerini bir araya getirerek ve gereksiz veri sürümlerini temizleyerek depolama alanını etkin bir şekilde kullanmaktadır. Cassandra, dağıtılmış doğası gereği birçok

Apache Cassandra, yüksek erişilebilirlik ve veri bütünlüğü sağlamak amacıyla gelişmiş hata toleransı mekanizmaları kullanmaktadır. Bu mekanizmalar, sistemdeki düğüm arızaları ve ağ kesintilerine karşı direnç sağlamak için tasarlanmıştır. Cassandra'nın hata toleransı sağlayan temel özellikleri şunlardır:

Veri Çoğaltma: Cassandra, verilerin hata toleransını artırmak için çoğaltma faktörü adı verilen bir parametre kullanmaktadır. Bu parametre, verinin kaç düğüme kopyalanacağını belirlemektedir. Veri çoğaltma stratejisi olarak Simple Strategy ve Network Topology Strategy olmak üzere iki ana yaklaşım sunmaktadır. Simple Strategy, veriyi sırayla düğümlere çoğaltırken, Network Topology Strategy ise veri merkezleri ve raf farkındalığını dikkate alarak çoğaltma işlemini yönetmektedir (Lerman & Ranganathan, 2016).

Tutarlılık Seviyeleri: Cassandra, yazma ve okuma işlemleri için farklı tutarlılık seviyeleri sunmaktadır. Bu seviyeler, işlemlerin başarılı sayılabilmesi için gerekli olan düğümlerden alınması gereken yanıtları belirlemektedir. Örneğin, QUORUM seviyesi, veri çoğaltma faktörüne bağlı olarak kümenin çoğunluğundan yanıt alındığında işlemin başarılı kabul edilmesini sağlamaktadır (Schroeder, 2015).

Otomatik Yükdevretme ve Veri Onarımı: İpucu Aktarımı mekanizması, geçici olarak ulaşılamayan bir düğüm için yazma işlemlerini diğer düğümlere geçici olarak saklamakta ve düğüm tekrar çevrimiçi olduğunda bu verileri geri göndermektedir. Okuma Onarımı ve Anti-Entropi Onarımı ise okuma sırasında veri tutarlılığını sağlamak ve düğümler arasında veri farklarını düzenli olarak senkronize etmek için kullanılmaktadır (Malik, 2015).

Veri Dengeleme ve Yeniden Dengeleme: Cassandra, yeni düğümler eklenip çıkarıldığında veya veri boyutlarında değişiklik olduğunda verilerin küme içinde

dengeli bir şekilde yeniden dağıtılmasını sağlamaktadır. Bu, sistemdeki performansı ve hata toleransını korumaktadır (Schroeder, 2020).

Bölge ve Veri Merkezi Farkındalığı: Cassandra, birden fazla veri merkezi arasında veri çoğaltarak hata toleransını artırmaktadır. Sütun farkındalığı ise aynı sütundaki düğümlerin arızalanması durumunda veri kaybını önlemektedir (Schroeder, 2020).

Kesintisiz Çalışma ve Ölçeklenebilirlik: Cassandra, dağıtık ve çoğaltılmış yapısıyla bakım işlemleri veya düğüm arızaları sırasında bile kesintisiz hizmet sunmaktadır. Yeni düğümler ekleyerek veya çıkararak sistem performansını ve kapasitesini çevrimiçi olarak ölçeklendirebilmektedir (Brebner, 2020).

Cassandra'nın bu hata toleransı mekanizmaları, büyük ölçekli ve kritik uygulamalar için güvenilir bir veritabanı çözümü sunmaktadır. Yüksek erişilebilirlik, veri bütünlüğü ve kolay ölçeklenebilirlik gibi özellikler, Cassandra'yı dağıtık ve yoğun iş yüklerine sahip uygulamalar için ideal hale getirmektedir (Malik, 2015).

3.10. HBase

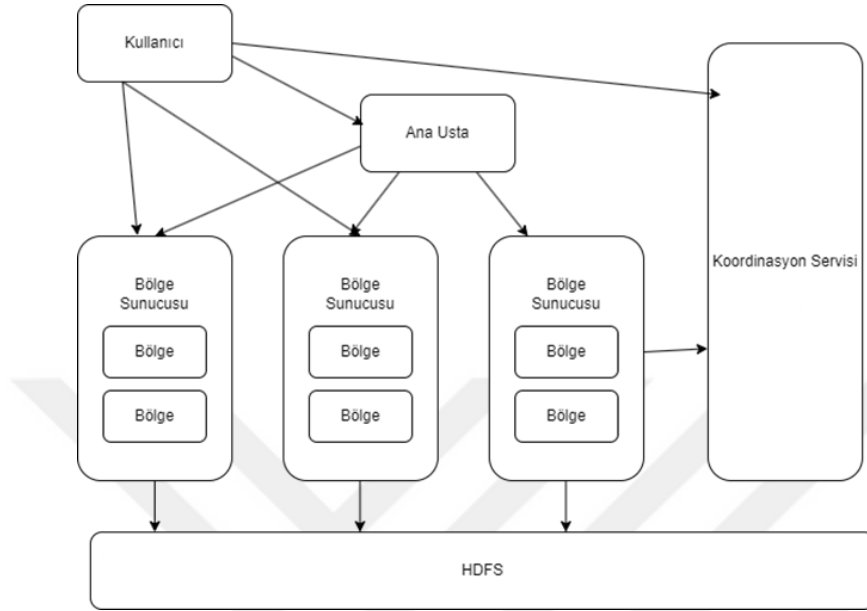
3.10.1. Mimari

HBase, Apache Hadoop ekosisteminin bir parçası olarak geliştirilen sütun odaklı ve dağıtık bir veritabanıdır. Özellikle büyük hacimli ve yapılandırılmış verilere hızlı erişim sağlamak amacıyla tasarlanmıştır. Google'ın büyük tablosu Bigtable'in açık kaynaklı bir uygulaması olarak düşünülebilir. HBase, Hadoop Dosya Sistemi (HDFS) üzerinde çalışmakta olup, HDFS'nin hata toleransını kullanarak verilerin güvenilir şekilde depolanmasını sağlamaktadır.

HBase'in temel özellikleri arasında yatay olarak ölçeklenebilir olması ve rastgele erişim gereksinimlerini karşılamak için optimize edilmiş olması yer almaktadır. Bu özellikleri sayesinde, büyük veri kümelerinde veri analizi, işlenmesi ve raporlanması gibi işlemler için etkin bir çözüm sunmaktadır. HBase, Hadoop Dosya Sistemi üzerinde depolanan verilere gerçek zamanlı olarak rastgele erişim sağlamakta ve bu sayede büyük ölçekli veri işleme işlerini desteklemektedir.

HBase'in kullanımı, özellikle büyük ölçekli veri yönetimi ve işleme gereksinimlerini karşılamak isteyen organizasyonlar için önemli bir seçenek sunmaktadır. Veri yoğun uygulamalar, zamanında veri analizi ve gerçek zamanlı karar destek sistemleri gibi senaryolarda HBase'in esnek ve güçlü mimarisi avantaj sağlamaktadır.

HBase, büyük veri yönetimi için güçlü bir araç olup, dağıtık yapısı ve HDFS ile entegrasyonu sayesinde verilerin güvenli, hızlı ve verimli bir şekilde işlenmesini sağlamaktadır. Bu özellikleri, HBase'i büyük ölçekli veri uygulamaları için ideal bir veritabanı çözümü haline getirmektedir (George, 2011).



Şekil 3.8. HBase Mimarisi.

3.10.2. Veri modeli

HBase, büyük veri kümeleri üzerinde çalışmak üzere tasarlanmış sütun odaklı bir NoSQL veritabanıdır ve Hadoop ekosisteminin önemli bir parçası olarak büyük ölçekli veri işleme ve depolama gereksinimlerini karşılamaktadır. HBase'in veri modeli, birçok bileşen içermekte olup bu bileşenler, verilerin organizasyonunu ve erişimini sağlamak için tasarlanmıştır.

HBase tabloları, verilerin temel organizasyon birimidir. Her tablo, satırlar ve sütun ailelerinden oluşmaktadır. Satırlar, benzersiz bir satır anahtarıyla tanımlanmakta ve bu anahtarlar üzerinden verilere erişim sağlanmaktadır. Her satır, aynı zamanda farklı zaman damgalarıyla saklanan farklı sürümleri içerebilmektedir. Varsayılan olarak, en yeni zaman damgasına sahip veri sürümü döndürülmektedir.

Sütun aileleri, fiziksel olarak birlikte depolanan sütun gruplarıdır ve bir HBase tablosunda bir veya daha fazla sütun ailesi bulunabilmektedir. Her sütun ailesi, ilişkili sütunların topluca yönetilmesini sağlamaktadır. Sütunlar ise dinamik olarak tanımlanabilmekte ve sütun ailesi adı ile sütun adı kombinasyonu ile belirtilmektedir.

Hücreler, HBase'de depolanan temel veri birimleridir. Her hücre, satır anahtarı, sütun ailesi adı, sütun adı ve zaman damgası kombinasyonu ile tekil olarak tanımlanmakta ve bu kombinasyonun içinde bir değer saklamaktadır. Bu yapı, HBase'in esnek ve ölçeklenebilir veri modelini oluşturan temel bileşenlerdir.

HBase'in bu veri modeli, büyük hacimli verilerin etkili bir şekilde saklanmasını, işlenmesini ve yönetilmesini sağlamaktadır. Özellikle, sütun odaklı yapısı ve esnek veri düzenleme imkanlarıyla büyük ölçekli dağıtılmış sistemler için ideal bir çözüm sunmaktadır (Taylor, 2016).

3.10.3. Yazma mekanizması

HBase, büyük veri kümelerinde tutarlılık, dayanıklılık ve performansı sağlamak amacıyla tasarlanmış sütun odaklı bir NoSQL veritabanıdır. Yazma işlemleri, dağıtılmış bir ortamda gerçekleştirilmekte ve bu işlem süreci, çeşitli temel mekanizmalar kullanılarak yönetilmektedir.

Yazma işlemleri ilk olarak "MemStore" adı verilen bellek tabanlı bir tampon alanında gerçekleştirilmektedir. MemStore, düğümlerdeki belleklerde bulunmakta ve gelen yazma istekleri burada hızla işlenmektedir. Bu yöntem, yazma işlemlerinin düşük gecikme süreleriyle tamamlanmasını sağlamaktadır.

Aynı zamanda, yazma işlemleri bir günlüğe (log) yazılmaktadır. Bu günlük, "Write-Ahead Log" (WAL) olarak adlandırılmakta ve HBase tablolarındaki tüm değişikliklerin güvenilir bir şekilde kaydedilmesini sağlamaktadır. WAL mekanizması, veri kaybını önlemekte ve veri bütünlüğünü sağlamaktadır.

MemStore'daki veriler, belirli bir boyuta veya zaman aralığına ulaştığında ya da belirli bir eşiğe ulaştığında, boşaltma işlemi ile disk üzerindeki kalıcı depolama alanına yazılmaktadır. Bu işlem, verilerin disk üzerindeki "HFile" adı verilen dosyalara kalıcı olarak yazılmasını sağlamaktadır.

Zamanla, disk üzerinde biriken HFile'lar, sıkıştırma işlemi ile birleştirilmekte ve sıkıştırılmaktadır. Sıkıştırma işlemi, disk alanının etkin bir şekilde kullanılmasını ve veriye erişimi optimize etmeyi amaçlamaktadır.

Bu mekanizmalar, HBase'in büyük veri kümelerinde güvenilir bir şekilde yazma işlemlerini yönetmesini sağlamaktadır. Tutarlılık, dayanıklılık ve performans gibi temel özelliklerle birlikte, HBase, dağıtılmış ve yüksek performanslı uygulamalar için ideal bir veri deposu çözümü sunmaktadır (George, 2011).

3.10.4. Okuma mekanizması

HBase, verilerin dağıtılmış bir şekilde depolandığı ve işlendiği ortamlarda tutarlılık, performans ve ölçeklenebilirlik sağlamak amacıyla tasarlanmış bir veritabanı sistemidir. HBase'in temel okuma mekanizması, "Bölge" adı verilen mantıksal veri bölgeleri üzerinde gerçekleştirilmektedir. Her bir Bölge, bir HBase tablosunun belirli bir anahtar aralığını temsil etmekte olup, tablonun büyümesi veya bölünmesi durumunda yeni Bölgeler oluşturulabilir veya mevcut Bölgeler bölünebilir ya da birleştirilebilmektedir.

Okuma işlemleri, öncelikle Bölge içindeki bellek tabanlı tampon olan "MemStore" üzerinden gerçekleştirilmektedir. Eğer istenen veri MemStore'da bulunmazsa, ardından disk üzerindeki kalıcı depolama alanında saklanan veri dosyaları olan "HFile"larda arama yapılmaktadır. Bu HFile'lar, HBase tablosundaki verilerin fiziksel olarak disk üzerinde depolandığı yapılandırmalardır.

Okuma işlemleri sırasında performansı artırmak için kullanılan bir diğer mekanizma ise "Ön Bellek"tir. Ön Bellek, sıkça erişilen verilerin bellek üzerinde tutulmasını sağlayarak, bu verilere daha hızlı erişim imkanı sunmaktadır. Bu sayede okuma işlemleri daha hızlı tamamlanabilmekte ve performans iyileştirilmektedir.

HBase'in bu okuma mekanizmaları, büyük veri kümelerinde verimli ve hızlı veri erişimi sağlamak amacıyla tasarlanmıştır. MemStore ve HFile gibi yapılar, verilerin organize bir şekilde depolanmasını ve hızlı bir şekilde erişilmesini mümkün kılarken, Ön Bellek gibi ek mekanizmalar performansın optimize edilmesine yardımcı olmaktadır. Bu özellikler, HBase'i büyük ölçekli ve yüksek performans gerektiren veri işleme uygulamaları için ideal bir çözüm haline getirmektedir (Edureka, 2022).

3.10.5. Sıkıştırma

HBase'de veri sıkıştırma, veri tabanındaki verilerin depolanması ve yönetilmesi süreçlerinde kullanılan önemli bir tekniktir. Bu teknik, depolama alanını optimize etmek ve veri tabanının performansını artırmak amacıyla tasarlanmıştır. HBase'in sıkıştırma mekanizması, blok sıkıştırması, sıkıştırma algoritmaları ve bölgesel sıkıştırma gibi temel bileşenleri içermektedir.

Blok sıkıştırması, genellikle HBase'in HFile dosyalarında uygulanan bir yöntemdir. HFile'lar, HBase tablosundaki verilerin disk üzerinde kalıcı olarak depolandığı dosyalardır. Blok sıkıştırması, her veri bloğunun içeriğini sıkıştırarak depolama

alanının daha etkin kullanılmasını sağlamaktadır. Bu sayede, aynı miktarda disk alanında daha fazla veri saklanabilmekte ve disk G/Ç işlemleri hızlanmaktadır.

HBase, çeşitli sıkıştırma algoritmalarını desteklemektedir. Bunlar arasında yaygın olarak kullanılan algoritmalar arasında Gzip, LZO ve Snappy gibi seçenekler bulunmaktadır. Bu algoritmalar, farklı sıkıştırma oranları ve işlem hızları sunmaktadır. Kullanılacak sıkıştırma algoritması, uygulamanın gereksinimlerine ve performans hedeflerine bağlı olarak seçilebilmektedir.

Ayrıca, HBase bölgesel sıkıştırma özelliğini desteklemektedir. Bu özellik, belirli bir HRegion içindeki tüm HFile'ların aynı sıkıştırma algoritmasıyla sıkıştırılmasını sağlamaktadır. Böylece, bir veri bölgesi disk üzerinde daha az yer kaplamakta ve disk G/Ç performansı artmaktadır.

HBase'in veri sıkıştırma mekanizmaları, büyük veri kümelerinin etkin bir şekilde yönetilmesini ve depolanmasını sağlamaktadır. Blok sıkıştırması, sıkıştırma algoritmaları ve bölgesel sıkıştırma gibi bileşenler, HBase'in veri depolama süreçlerini optimize ederek performansın artırılmasına katkıda bulunmaktadır. Bu özellikler, HBase'i büyük ölçekli veri yönetimi ve yüksek performans gereksinimlerine sahip uygulamalar için ideal bir çözüm haline getirmektedir (Risden, 2018).



4. TEST ORTAMI VE KONFIGÜRASYON

4.1. Küme Özelliği

Çalışmanın amacı doğrultusunda MongoDB, Couchbase, Cassandra ve HBase veritabanlarının donanım üzerinde gerçekleştirdiği iş çıkarma oranı, gecikme süresi, işlemci kullanımı, bellek kullanımı ve iş parçacığı ve iş yüküne göre performansları gözlemlenmiştir.

Bu bağlamda her bir sunucu için 12 çekirdekli işlemci, 16 GB bellek, Oracle Linux 8.7 işletim sistemi içeren toplamda 3 sunucu kurulumların yapılmış ve performans ölçümlerinin gerçekleştirilmiştir. Kurulumların yapılması için kullanılacak donanım özellikleri tablo 4.1’de belirtilmiştir.

Tablo 4.1. Test kapsamındaki sunucu özelliği.

Gereksinim	Özellik
İşlemci	Intel Xeon 6230R
Bellek	16 GB Ram
Çekirdek	12 Cores
İşletim Sistemi	Oracle Linux 8.7

4.2. YCSB

Çalışmanın amacı doğrultusunda MongoDB, Couchbase, Cassandra ve HBase veritabanlarının donanım üzerinde gerçekleştirdiği iş çıkarma oranı, gecikme süresi, işlemci kullanımı, bellek kullanımı ve iş parçacığı ve iş yüküne göre performansları gözlemlenmiştir.

YCSB, veritabanı sistemlerinin performansını objektif bir şekilde karşılaştırabilme ve sistemlerin belirli kullanım durumları için en uygun olduğunu belirleme konusunda araştırmacılar ve endüstri profesyonelleri için önemli bir araçtır. YCSB, farklı

veritabanı sistemlerinin performansını ölçmek ve karşılaştırmak amacıyla geliştirilmiş açık kaynaklı bir performans test aracıdır. Özellikle NoSQL veritabanları üzerinde odaklanan bu araç, veritabanlarının okuma, yazma, güncelleme ve silme gibi temel işlemlerini değerlendirir ve performanslarını metrikler aracılığıyla ölçmektedir.

YCSB'nin temel amacı, önceden tanımlanmış iş yükleri aracılığıyla çeşitli veri işleme senaryolarını simüle etmektir. Her iş yükü, belirli bir kullanım durumunu temsil eder ve bu durumlar, farklı veri işlemlerini içermektedir. Varsayılan olarak YCSB, altı farklı iş yükü sunmakta ve her biri farklı performans testi senaryosu sağlamaktadır.

YCSB'nin kullanımıyla elde edilen performans metrikleri arasında gecikme süresi ve işlem hacmi bulunmaktadır. Araç, farklı veritabanlarıyla entegre olabilmesi için genişletilebilir bir yapıya sahiptir ve kullanıcıların kendi ihtiyaçlarına göre iş yüklerini özelleştirmesine olanak tanımaktadır. Bu özellikleriyle YCSB, veritabanı karşılaştırmaları, sistem optimizasyonu ve ölçeklenebilirlik testleri gibi alanlarda yaygın olarak kullanılmaktadır.

YCSB'nin performans testlerinde kullanılan veri dağılımlarından biri Zipfian ve diğeri Uniform'dur. Zipfian dağılımı, gerçek dünya senaryolarını daha iyi taklit etmekte ve bazı verilerin diğerlerinden çok daha sık erişildiği durumları yansıtmaktadır. Örneğin, web sitelerinde popüler sayfaların nadiren ziyaret edilen sayfalardan daha sık ziyaret edildiği bir durumu simgelemektedir. Bu dağılım, gerçek dünya uygulamalarında sıkça karşılaşılan bir durumu modellemek için kullanılmaktadır.

Diğer yandan, Uniform dağılımı ise verilerin eşit olarak dağıldığı ideal bir senaryoyu simgelemektedir. Tüm veri noktalarının aynı sıklıkta erişildiği varsayılmakta, bu da bazı özel test senaryoları için kullanışlı olmaktadır. Ancak, genellikle gerçek dünya senaryolarını yansıtmak için Zipfian dağılımı tercih edilmektedir.

YCSB, veritabanı sistemlerinin performansını objektif bir şekilde karşılaştırabilme ve sistemlerin belirli kullanım durumları için en uygun olduğunu belirleme konusunda araştırmacılar ve endüstri profesyonelleri için önemli bir araçtır. Bu araç, veritabanı sistemlerinin performansını değerlendirmek, karşılaştırmak ve optimize etmek için yaygın olarak kullanılmaktadır (Cooper ve ark., 2010).

4.3. MongoDB Konfigürasyonu

Çalışmanın amacı doğrultusunda MongoDB, Couchbase, Cassandra ve HBase veritabanlarının donanım üzerinde gerçekleştirdiği iş çıkarma oranı, gecikme süresi, işlemci kullanımı, bellek kullanımı ve iş parçacığı ve iş yüküne göre performansları gözlemlenmiştir.

Bu çalışmada, MongoDB'nin replika set özelliği kullanılarak veritabanı sisteminin dayanıklılığını ve performansını değerlendirdik.

- **Replika Seti Oluşturma**

Her bir replika set düğümü için aşağıdaki adımlar takip edilmektedir.

- **Data Path Oluşturma:** Her düğüm için bir data path belirlenmekte ve oluşturulmaktadır. Bu adım, her düğümün kendi veri depolama alanını oluşturmasını sağlamaktadır.

```
1 mkdir -p /data/node1
2 mkdir -p /data/node2
3 mkdir -p /data/node3
```

Şekil 4.1. Veri Yolu Komutları.

- **MongoDB Başlatma:** MongoDB sunucusu, belirtilen data path ile başlatılmaktadır.

```
1 ./mongod --MongoCluster mongoTest --port 27017 --dbpath /data/node1
```

Şekil 4.2. MongoDB Başlatma Komutu.

Diğer düğümler için de aynı komut kullanılarak başlatma işlemi gerçekleştirilmektedir.

- **Replika Seti Konfigürasyonu**

Replika seti oluşturmak için aşağıdaki adımlar takip edilmektedir.

rs.initiate() Komutu: Birincil düğümü başlatmak için replika seti başlatılmaktadır. Bu adım, replika seti yapılandırmasını ve birincil düğümü başlatmaktadır.

- **Konfigürasyon Ayarları:** Replika setinin yapılandırması belirlenmektedir. Bu yapılandırma, replika setinin adını belirlemekte ve her düğümün kimliklerini ve bağlantı bilgilerini içermektedir.

```
1 config = {  
2   _id: 'mongoTest',  
3   members: [  
4     { _id: 0, host: 'localhost:27017' }, // Birincil düğüm  
5     { _id: 1, host: 'node2_ip:27017' }, // İkincil düğüm  
6     { _id: 2, host: 'node3_ip:27017' }  // Üçüncül düğüm  
7   ]  
8 }
```

Şekil 4.3. Konfigürasyon Komutları.

- **rs.add() Komutu:** İkincil düğümleri replika setine eklemek için aşağıdaki komutlar kullanılmaktadır.

```
1 rs.add("node2_ip:27017")  
2 rs.add("node3_ip:27017")
```

Şekil 4.4. rs.add() Komutları.

Bu komutlar, replika setine yeni düğümleri ekler ve veritabanı sisteminin yüksek erişilebilirliğini sağlamaktadır.

4.4. Couchbase Konfigürasyonu

Bu çalışmada, Couchbase 7.6.1 sürümü kullanılarak bir küme oluşturulması ve yönetilmesi süreci anlatılmaktadır.

- **Cluster Kurulumu**

İlk adım olarak, birincil düğüm üzerinde Couchbase kümesini başlatmak için aşağıdaki komut kullanılmaktadır.

```
1 couchbase-cli cluster-init -c 127.0.0.1:8091  
2 --cluster-username <admin_user> --cluster-password <admin_password>  
3 --services data,index,query --cluster-ramsize 6000  
4 --cluster-index-ramsize 512 --cluster-fts-ramsize 256
```

Şekil 4.5. Couchbase Başlatma Komutları.

Bu komut, Couchbase kümesini başlatırken gerekli yapılandırmaları yapmaktadır.

İkinci ve üçüncü düğümler, oluşturulan kümeye eklenmektedir. Bu işlem için aşağıdaki komutlar kullanılmaktadır.

```
1 couchbase-cli server-add -c <first_node_ip>:8091 --username <admin_user>
2 --password <admin_password> --server-add <node_ip>:8091
3 --server-add-username <admin_user> --server-add-password <admin_password>
4 --services data,index,query
```

Şekil 4.6. Düğüm Ekleme Komutu.

Bu komutlar, mevcut kümelere yeni düğümlerin eklenmesini sağlamaktadır.

- **Dengeleme İşlemi**

Düğümlerin eklenmesi işleminden sonra, kümenin dengelenmesi yapılmaktadır. Bu adım, verilerin yeni eklenen düğümlere dağıtılmasını sağlamaktadır.

```
1 couchbase-cli rebalance -c <first_node_ip>:8091
2 --username <admin_user> --password <admin_password>
```

Şekil 4.7. Dengeleme Komutu.

4.5. Cassandra Konfigürasyonu

Bu çalışmada, Cassandra 4.1.5 sürümü kullanılarak yapılan küme konfigürasyonu ve test süreci adımları detaylı olarak anlatılmaktadır.

- **Yaml Dosyası Yapılandırması**

İlk adım olarak, Cassandra kümesini yapılandırmak için kullanılan yaml dosyasının belirli parametreleri ayarlanmaktadır.

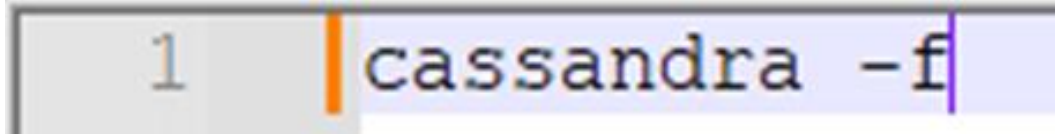
```
1 cluster_name: 'Test Cluster'
2 listen_address: <node_ip>
3 seeds: "<seed_node1_ip>,<seed_node2_ip>"
```

Şekil 4.8. Yaml Konfigürasyon Komutu.

- **Cassandra Küme Kurulumu ve Yapılandırma**

Cassandra kümesini oluşturmak için aşağıdaki adımlar takip edilmektedir:

- **Küme İnisiyalizasyonu:** İlk düğümde aşağıdaki komut ile Cassandra kümesi başlatılmaktadır.



Şekil 4.9. Cassandra Başlatma Komutu.

- **Düğümlerin Eklenmesi:** Diğer düğümler kümeye eklenmekte ve seed düğümleri belirtilen IP adresleri ile yapılandırılmaktadır. Bu adım, ağda daha fazla düğüm ekleyerek kümenin ölçeklenmesini sağlamaktadır.

4.6. HBase Konfigürasyonu

Bu çalışmada, Hadoop 3.4.0 ve HBase 2.6.0 sürümleri kullanılarak yapılan kurulum ve yapılandırma işlemleri adım adım açıklanmaktadır.

- **Hadoop Kurulumu**

İlk adım olarak, Hadoop 3.4.0 sürümü kurulmaktadır. Hadoop, büyük veri dosyalarını depolamak ve paralel olarak işlemek için kullanılan temel bir bileşendir. Kurulum sürecinde şu adımlar takip edilmektedir:

- Hadoop'un resmi web sitesinden veya uygun bir depodan kurulum paketleri indirilmektedir.
- Gerekli bağımlılıklar ve sistem gereksinimleri sağlanmaktadır.
- Yapılandırma dosyaları (örneğin, core-site.xml, hdfs-site.xml) üzerinde gerekli ayarlamalar yapılmaktadır.

- **HBase Kurulumu**

Hadoop kurulumundan sonra, HBase 2.6.0 sürümü kurulmaktadır. HBase, Hadoop üzerinde çalışan, yüksek performanslı ve büyük ölçekli veri tabanları için optimize edilmiş bir çözümdür. Kurulum sürecinde şu adımlar izlenmektedir:

- HBase'nin resmi web sitesinden veya uygun bir depodan kurulum paketleri indirilmektedir.
- Gerekli bağımlılıklar ve Hadoop ile uyumlu sürümler kontrol edilmektedir.
- HBase yapılandırma dosyası olan hbase-site.xml üzerinde gerekli

ayarlamalar yapılmaktadır. Bu yapılandırma, HBase'nin Hadoop üzerinde çalışacağı dosya sistemi kök dizinini (hbase.rootdir) ve ZooKeeper kümesinin IP adreslerini (hbase.zookeeper.quorum) belirtmektedir.

```
1 <configuration>
2   <property>
3     <name>hbase.rootdir</name>
4     <value>hdfs://<namenode_ip>:9000/hbase</value>
5   </property>
6   <property>
7     <name>hbase.zookeeper.quorum</name>
8     <value><node1_ip>,<node2_ip>,<node3_ip></value>
9   </property>
10 </configuration>
```

Şekil 4.10. Düğüm Yapılandırma Komutu.



5. ARAŞTIRMA BULGULARI

Veritabanı sistemlerinin performans değerlendirmeleri, çeşitli kullanım senaryolarını yansıtan farklı iş yükleri ile gerçekleştirilmektedir. Her senaryo, belirli bir iş yükü türünü ve kullanım amacını temsil etmektedir. Bu çalışma, altı farklı iş yükü senaryosunu incelemekte ve her birinin performans değerlendirmelerini tartışmaktadır.

İlk olarak, Okuma-Yazma Karışık (İş Yükü A) senaryosu ele alınmaktadır. Bu senaryo, %50 okuma ve %50 güncelleme işlemlerini içermekte olup, sosyal ağlar gibi güncelleme-ağırlıklı senaryolar için kullanılmaktadır. Bu tür senaryolarda, veritabanının hem okuma hem de güncelleme işlemlerinde gösterdiği performans değerlendirilmektedir. Sosyal medya platformlarında, kullanıcıların içerik görüntüleme ve güncelleme işlemleri simüle edilmektedir.

İkinci olarak, Okuma-Ağırlıklı (İş Yükü B) senaryosu incelenmektedir. Bu senaryo, %95 okuma ve %5 güncelleme işlemlerini içermekte olup, profil aramaları gibi okuma-ağırlıklı uygulamalar için kullanılmaktadır. Bu tür senaryolarda, veritabanının okuma işlemlerinde ne kadar etkili olduğu değerlendirilmektedir. Profil arama uygulamalarında, kullanıcıların çoğunlukla içerik tarama ve okuma işlemleri gerçekleştirdiği göz önüne alındığında, okuma işlemlerinin performansı kritik bir öneme sahiptir.

Üçüncü olarak, Sadece Okuma (İş Yükü C) senaryosu ele alınmaktadır. Bu senaryo, %100 okuma işlemlerini içermekte olup, yalnızca okuma gerektiren uygulamalar, örneğin önbellekler, için kullanılmaktadır. Bu tür senaryolarda, veritabanının okuma performansı test edilmekte ve genellikle en kritik performans göstergesi olarak kabul edilmektedir.

Dördüncü olarak, Okuma ve Ekleme (İş Yükü D) senaryosu incelenmektedir. Bu senaryo, %75 okuma ve %25 ekleme işlemlerini içermekte olup, kullanıcı kaydı senaryoları için kullanılmaktadır. Özellikle yeni kullanıcıların sürekli eklendiği

sistemlerde, veritabanının kullanıcı bilgilerini okuma ve yeni kullanıcıları ekleme işlemlerindeki performansı değerlendirilmektedir.

Beşinci olarak, Tarama (İş Yüğü E) senaryosu ele alınmaktadır. Bu senaryo, %75 ekleme ve %25 güncelleme işlemlerini içermekte olup, analitik uygulamalar, örneğin raporlama ve veri analizi, için kullanılmaktadır. Bu tür senaryolarda, veri taramalarının sıklığı ve veritabanına yeni veri eklemeleri arasındaki güncellemelerin performansı değerlendirilmektedir.

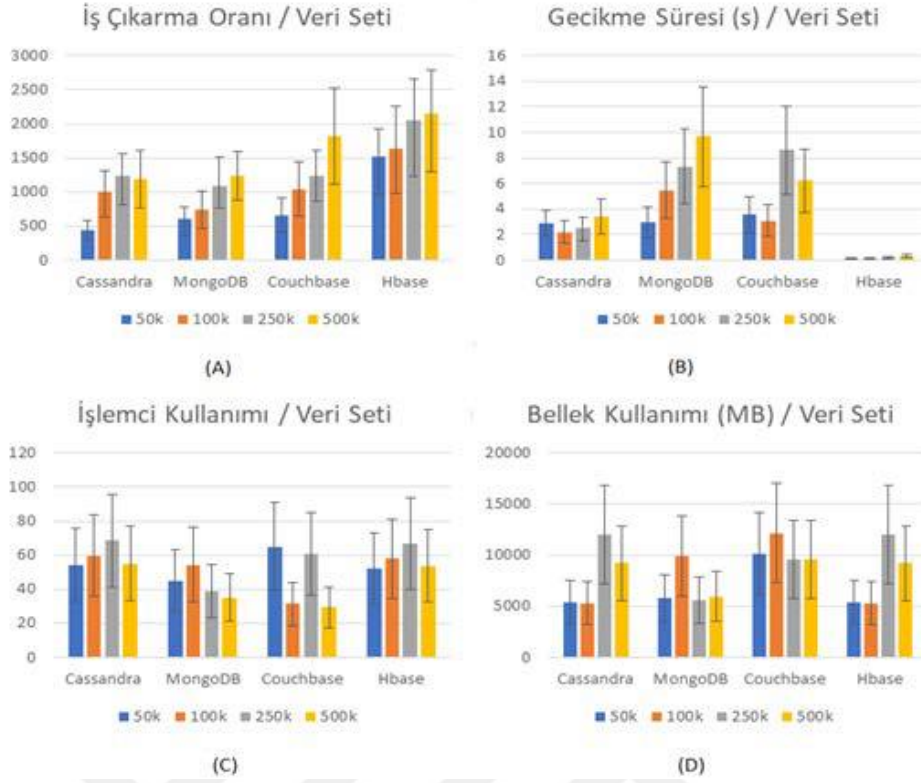
Son olarak, Okuma-Değiştirme (İş Yüğü F) senaryosu incelenmektedir. Bu senaryo, %50 okuma ve %50 okuma-güncelleme-yazma işlemlerini içermekte olup, kullanıcı profili güncellemeleri gibi kayıt güncelleme senaryoları için kullanılmaktadır. Bu tür senaryolarda, kullanıcı profilleri okuma, güncelleme ve yazma işlemlerine ilişkin karışık iş yükleri simüle edilmekte ve veritabanının performansı değerlendirilmektedir.

Her iş yükü senaryosu, belirli bir kullanım durumunu ve veritabanı performansının farklı yönlerini test etmek amacıyla tasarlanmaktadır. Veritabanı sistemlerinin performansını anlamak ve iyileştirmek için bu tür çeşitli senaryoların kullanılması önem arz etmektedir. İş yükü senaryolarının seçimi, uygulamanın gerçek dünyadaki kullanımına ve performans gereksinimlerine uygun olmalıdır.

5.1. İş Yüğü A

5.1.1. Test – 1

%50 okuma %50 güncelleme üzerinde oluşturulan test ortamında 50K, 100K, 250K ve 500K verisetindeki iş çıktı oranı, gecikme süresi, işlemci kullanımı ve bellek kullanımı parametreleriyle testler gerçekleştirilmiştir. Şekil 5.1’de yapılan testlerden elde edilen sonuçlar görselleştirilmiştir.



Şekil 5.1. Test – 1 Sonucu.

YCSB aracı ile 50K, 100K, 250K ve 500K üzerinde yapılan iş çıkarma, gecikme süresi, işlemci kullanımı ve bellek kullanımı test sonuçları incelendiğinde;

Şekil 5.1(A), dört NoSQL veritabanı sisteminin iş çıkarma oranlarını göstermektedir. HBase, tüm veri seti boyutlarında en yüksek iş çıkarma oranına ulaşarak üstün bir performans sergilemiştir. MongoDB, veri seti boyutları arttıkça iş çıkarma oranında artış göstermekte ve her boyutta tutarlı bir şekilde yüksek performans sergilemektedir. Couchbase de veri seti boyutları arttıkça iş çıkarma oranında artış göstermiş ve 500k veri setinde yüksek bir performans sergilemiştir. Cassandra ise genel olarak daha düşük iş çıkarma oranları sergilemiş, özellikle küçük veri setlerinde düşük performans göstermiştir. Cassandra'nın 250k veri seti boyutunda performansı artmış, ancak 500k veri seti boyutunda bir düşüş gözlemlenmiştir.

Şekil 5.1(B), dört NoSQL veritabanının gecikme sürelerini göstermektedir. Cassandra, 50k veri setinde yüksek bir gecikme süresine sahipken, 100k veri setinde gecikme süresi düşmektedir. 250k veri setinde gecikme süresi tekrar artmakta ve en yüksek gecikme süresi 500k veri setinde gözlemlenmektedir. MongoDB, 50k ve 100k veri setlerinde bir miktar düşük gecikme sürelerine sahipken, 250k ve 500k veri setlerinde gecikme süresi belirgin bir şekilde artmaktadır. Couchbase, özellikle 100k veri seti

boyutunda düşük gecikme süresine sahipken, 250k ve 500k veri seti boyutlarında gecikme süresi önemli ölçüde artmaktadır. HBase, tüm veri seti boyutlarında en düşük gecikme sürelerine sahip olup, performans açısından üstünlüğünü korumaktadır.

Şekil 5.1(C), dört NoSQL veritabanının işlemci kullanımını göstermektedir. Cassandra, veri seti boyutları arttıkça CPU kullanımında belirgin bir artış sergilemekte, özellikle 250k veri seti boyutunda en yüksek kullanım oranına ulaşmaktadır. MongoDB, 50k ve 100k veri setlerinde yüksek CPU kullanımı gösterirken, 250k veri setinde belirgin bir düşüş ve 500k veri setinde en düşük seviyede CPU kullanımı gözlemlenmektedir. Couchbase, 50k veri seti boyutunda en yüksek CPU kullanım oranına sahip olup, 100k veri setinde düşük bir kullanım göstermektedir. Ancak, 250k veri setinde tekrar yüksek bir kullanım sergilemekte ve 500k veri setinde CPU kullanımı önemli ölçüde düşmektedir. HBase, 50k veri seti boyutunda nispeten düşük CPU kullanımına sahipken, 100k veri setinde bu kullanım artmakta, 250k veri setinde en yüksek seviyeye ulaşmakta ve 500k veri setinde ise daha düşük bir kullanım oranı göstermektedir.

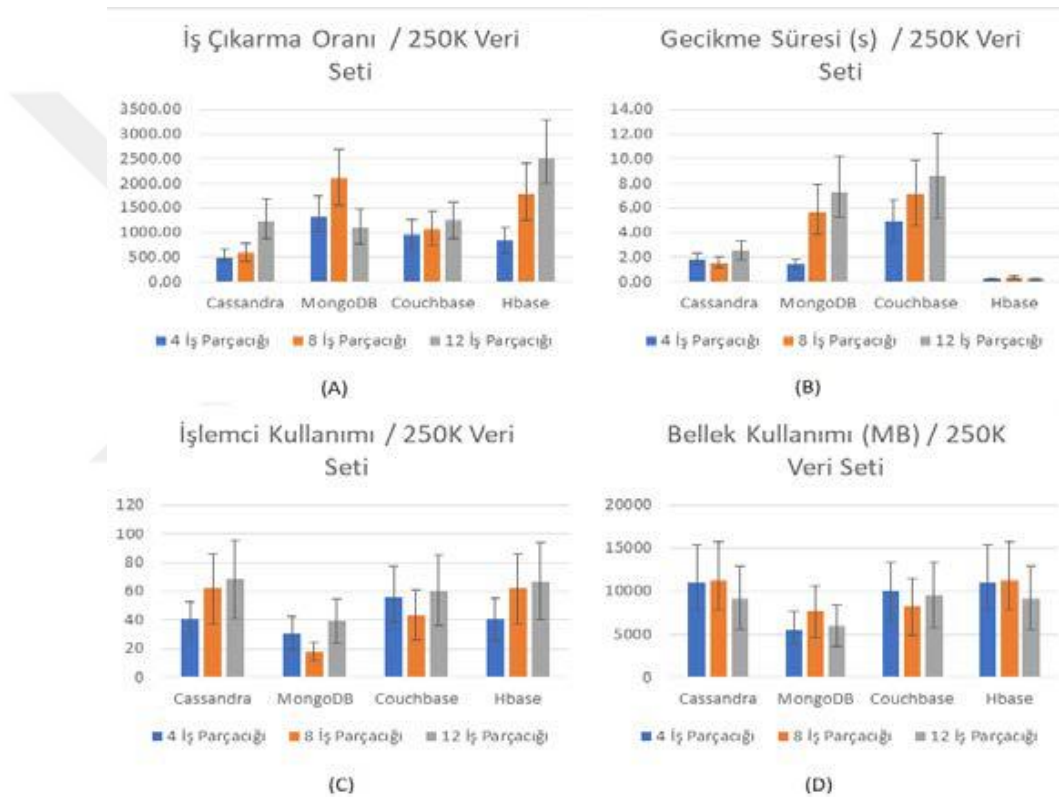
Şekil 5.1(D), dört NoSQL veritabanının bellek kullanım oranlarını göstermektedir. Cassandra, 250k veri seti boyutunda en yüksek bellek kullanımına ulaşmakta, diğer veri seti boyutlarında ise daha düşük ve tutarlı bir bellek kullanımı sergilemektedir. MongoDB, 100k veri seti boyutunda en yüksek bellek kullanımına sahipken, diğer veri seti boyutlarında bellek kullanımı birbirlerine yakın seviyelerde gözlemlenmektedir. Couchbase, 100k veri seti boyutunda en yüksek bellek kullanımına sahiptir ve 50k veri setinde de belirgin derecede yüksek bellek kullanımı göstermektedir. Ancak, 250k ve 500k veri seti boyutlarında bellek kullanımı nispeten daha düşüktür. HBase, 250k veri seti boyutunda en yüksek bellek kullanımını sergilemekte, 50k ve 100k veri seti boyutlarında ise daha düşük ve dengeli bir bellek kullanımı göstermektedir.

HBase, dağıtık ve ölçeklenebilir yapısıyla iş çıkarma oranında üstün performans sergilemektedir. Bu durum, HBase'in veri replikasyonu ve dağıtık dosya sistemi sayesinde veriyi paralel olarak işleyebilmesinden kaynaklanmaktadır. MongoDB, esnek veri modeli ve güçlü indeksleme mekanizmaları ile tutarlı performans göstermektedir. Couchbase, otomatik yük devretme ve veri yeniden dengeleme özellikleri sayesinde yüksek erişilebilirlik ve veri tutarlılığı alanlarında desteklenmektedir. Cassandra, küçük veri setlerinde yetersiz optimizasyonlardan dolayı iş çıkarma oranında daha düşük performans göstermektedir.

Veri boyutu ve bellek kullanımı arasında genel olarak bir ilişki bulunmakla birlikte, bu ilişki her zaman doğrusal değildir. İş yüklerine göre performans değerlendirmesi yapıldığında, veri boyutunun artması her zaman performansı tam anlamıyla iyileştirmemektedir. Daha iyi sonuçlar elde etmek için, her veritabanının konfigürasyon ayarları ve veri işleme stratejileri optimize edilmelidir.

5.1.2. Test – 2

%50 okuma %50 güncelleme üzerinde oluşturulan test ortamında 250K verisetinde 4, 8 ve 12 iş parçacığı kullanılarak testler gerçekleştirilmiştir. Şekil 5.2’de yapılan testlerden elde edilen sonuçlar görselleştirilmiştir.



Şekil 5.2. Test – 2 Sonucu.

Şekil 5.2(A), farklı iş parçacığı sayılarına göre 250k veri seti boyutunda iş çıkarma oranlarını göstermektedir. Cassandra, iş parçacığı sayısı arttıkça iş çıkarma oranında kademeli bir artış göstermekte, ancak bu artış diğer veritabanlarına kıyasla daha sınırlı kalmaktadır. MongoDB, 8 iş parçacığı ile en yüksek iş çıkarma oranına ulaşmakta ve 4 iş parçacığı ile de yüksek performans sergilemektedir. 12 iş parçacığı ile ise performans biraz düşmektedir. Couchbase, iş parçacığı sayısına bağlı olarak iş çıkarma oranında değişiklik göstermekte, 12 iş parçacığı ile en yüksek performansı sergilemektedir. HBase, 12 iş parçacığı ile en yüksek iş çıkarma oranına sahip olup, 8

iş parçacığı ile de yüksek bir performans sergilemektedir. 4 iş parçacığı ile nispeten daha düşük bir iş çıkarma oranına sahiptir.

Şekil 5.2(B), farklı iş parçacığı sayılarına göre 250k veri seti boyutunda gecikme sürelerini göstermektedir. Cassandra, 4 ve 8 iş parçacığında nispeten düşük gecikme sürelerine sahipken, 12 iş parçacığında gecikme süresi biraz artmaktadır. MongoDB, 4 iş parçacığında düşük bir gecikme süresi sergilerken, 8 iş parçacığında gecikme süresi önemli ölçüde artmakta ve 12 iş parçacığında en yüksek seviyeye ulaşmaktadır. Couchbase, iş parçacığı sayısı arttıkça kademeli olarak artan gecikme sürelerine sahiptir ve 12 iş parçacığında en yüksek gecikme süresine ulaşmaktadır. HBase, tüm iş parçacığı sayılarında en düşük gecikme sürelerine sahip olup, performans açısından diğer veritabanlarına göre üstünlük göstermektedir.

Şekil 5.2(C), farklı iş parçacığı sayılarına göre 250k veri seti boyutunda işlemci kullanım oranlarını göstermektedir. Cassandra, iş parçacığı sayısı arttıkça işlemci kullanımında belirgin bir artış sergilemekte ve 12 iş parçacığında en yüksek seviyeye ulaşmaktadır. MongoDB, 4 iş parçacığında yüksek işlemci kullanımı gösterirken, 8 iş parçacığında bu oran düşmekte ve 12 iş parçacığında en yüksek seviyeye ulaşmaktadır. Couchbase, 4 iş parçacığında yüksek işlemci kullanımı sergilemekte, 8 iş parçacığında işlemci kullanımı azalmakta ve 12 iş parçacığında en yüksek seviyeye ulaşmaktadır. HBase, 4 ve 8 iş parçacığında yüksek işlemci kullanımı sergilerken, 12 iş parçacığında bu oran en yüksek seviyeye çıkmaktadır.

Şekil 5.2(D), farklı iş parçacığı sayılarına göre 250k veri seti boyutunda bellek kullanım oranlarını göstermektedir. Cassandra, 4 ve 8 iş parçacığında yüksek bellek kullanımı sergilemekte, 12 iş parçacığında ise nispeten daha düşük bir bellek kullanımı gözlemlenmektedir. MongoDB, 4 iş parçacığında en düşük bellek kullanımı gösterirken, 8 iş parçacığında bu kullanım artmakta ve 12 iş parçacığında tekrar düşmektedir. Couchbase, 4 iş parçacığında yüksek bellek kullanımı sergilemekte, 8 iş parçacığında bellek kullanımı azalmakta ve 12 iş parçacığında tekrar yükselmektedir. HBase ise 4 ve 8 iş parçacığında yüksek bellek kullanımı gösterirken, 12 iş parçacığında bu oran biraz daha düşmektedir.

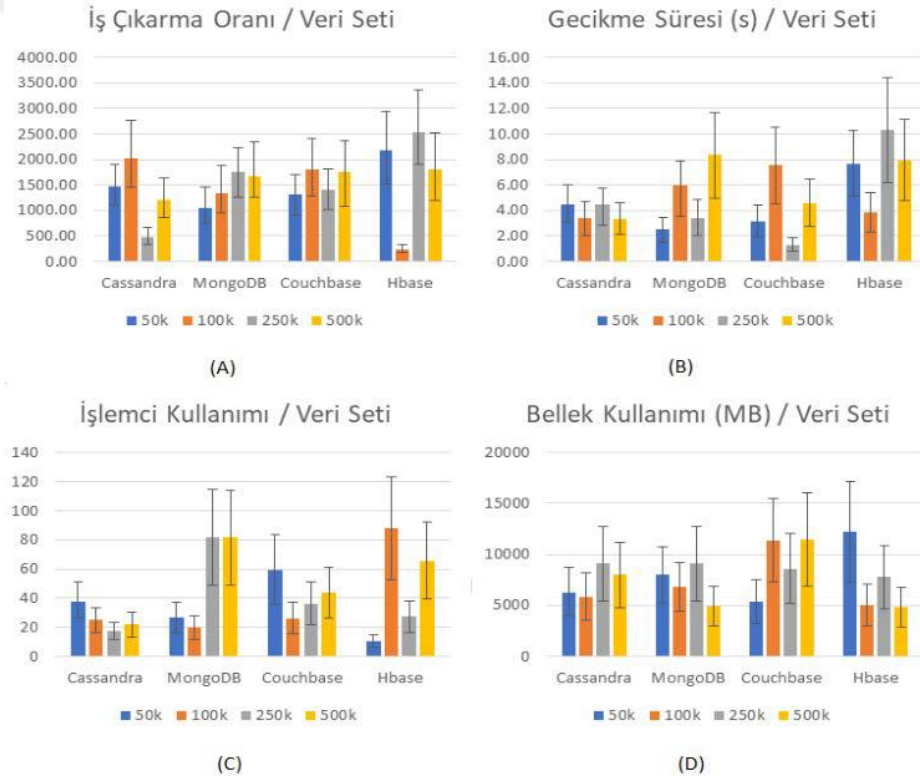
HBase, iş parçacığı sayısı arttıkça yüksek iş çıkarma oranı ve düşük gecikme süreleri ile üstün performans sergilemektedir. Bu durum, HBase'in veri replikasyonu ve dağıtık dosya sistemi sayesinde veriyi paralel olarak işleyebilmesinden kaynaklanmaktadır.

MongoDB, esnek veri modeli ve güçlü indeksleme mekanizmaları ile 8 iş parçacığında en yüksek iş çıkarma performansı göstermektedir, ancak 12 iş parçacığında performans düşmektedir. Couchbase, otomatik yükdevretme ve veri yeniden dengeleme özellikleriyle iş parçacığı sayısı arttıkça yüksek iş çıkarma performans sergilerken, Cassandra iş parçacığı sayısı arttıkça işlemci kullanımında belirgin bir artış göstermesine rağmen sınırlı bir iş çıkarma performans artışı sunmaktadır. İş parçacığı sayısı ve bellek kullanımı arasında genel olarak bir ilişki bulunmakla birlikte, bu ilişki her zaman doğrusal değildir. İş parçacığı sayısındaki artış, iş çıkarma oranında kısmen paralellik göstermektedir. Daha iyi sonuçlar elde etmek için, her veritabanının konfigürasyon ayarları ve veri işleme stratejileri optimize edilmelidir.

5.2. İş Yüğü B

5.2.1. Test – 1

%95 okuma %5 güncelleme üzerinde oluşturulan test ortamında 50K, 100K, 250K ve 500K verisetindeki iş çıktı oranı, gecikme süresi, işlemci kullanımı ve bellek kullanımı parametreleriyle testler gerçekleştirilmiştir. Şekil 5.3’de yapılan testlerden elde edilen sonuçlar görselleştirilmiştir.



Şekil 5.3. Test – 1 Sonucu.

Şekil 5.3(A), dört NoSQL veritabanının farklı veri seti boyutlarına göre iş çıkarma oranlarını göstermektedir. Cassandra, 50k ve 100k veri setlerinde yüksek iş çıkarma oranlarına sahipken, 250k veri setinde bu oran önemli ölçüde düşmektedir ve 500k veri setinde tekrar artmaktadır. MongoDB, veri seti boyutları arttıkça iş çıkarma oranında genellikle artış göstermekte olup, özellikle 250k veri setinde en yüksek oranına ulaşmaktadır. Couchbase, 50k veri setinde düşük bir iş çıkarma oranına sahipken, 100k veri setinde en yüksek performansa ulaşmakta, 250k veri setinde tekrar düşmekte ve 500k veri setinde yeniden artmaktadır. HBase, 50k veri setinde düşük iş çıkarma oranına sahipken, 250k veri setinde en yüksek performansa ulaşmakta ve 500k veri setinde de yüksek bir oran göstermektedir.

Şekil 5.3(B), dört NoSQL veritabanının farklı veri seti boyutlarına göre gecikme sürelerini göstermektedir. Cassandra, 50k veri setinde en yüksek gecikme süresine sahipken, 100k veri setinde düşmekte, 250k veri setinde artmakta ve 500k veri setinde yeniden düşmektedir. MongoDB, 50k ve 250k veri setlerinde düşük gecikme süreleri sergilemekte, 100k ve 500k veri setlerinde ise gecikme süresi artmaktadır. Couchbase, 50k veri setinde düşük gecikme süresi göstermekte, 100k veri setinde artmakta, 250k veri setinde düşmekte ve 500k veri setinde yeniden artmaktadır. HBase, 100k veri setinde en düşük gecikme süresine sahipken, 50k veri setinde artmakta ve 250k ile 500k veri setlerinde yüksek gecikme süreleri göstermektedir.

Şekil 5.3(C), dört NoSQL veritabanının farklı veri seti boyutlarına göre işlemci kullanım oranlarını göstermektedir. Cassandra, 50k veri setinde yüksek işlemci kullanımı sergilemekte, 100k ve 250k veri setlerinde bu kullanım düşmekte ve 500k veri setinde yeniden artmaktadır. MongoDB, 250k ve 500k veri setlerinde yüksek işlemci kullanım oranlarına ulaşmakta, 50k ve 100k veri setlerinde ise daha düşük oranlar göstermektedir. Couchbase, 50k veri setinde yüksek işlemci kullanımı sergilemekte, 100k veri setinde düşmekte, 250k veri setinde tekrar artış göstermekte ve 500k veri setinde artmaya devam etmektedir. HBase, 50k veri setinde düşük işlemci kullanımı sergilemekte, 100k veri setinde önemli ölçüde artmakta, 250k veri setinde düşmekte ve 500k veri setinde tekrar yüksek seviyelere çıkmaktadır.

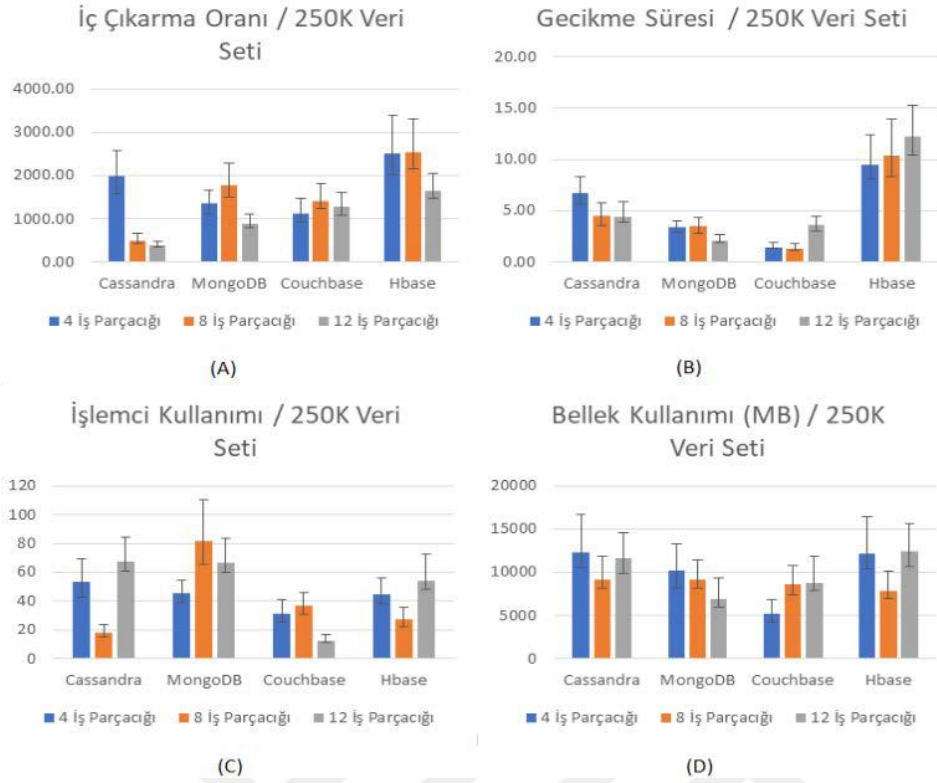
Şekil 5.3(D), dört NoSQL veritabanının farklı veri seti boyutlarına göre bellek kullanım oranlarını göstermektedir. Cassandra, 50k veri setinde yüksek bellek kullanımı sergilemekte, 100k veri setinde bu kullanım biraz düşmekte, 250k veri setinde önemli ölçüde artmakta ve 500k veri setinde tekrar düşmektedir. MongoDB,

50k veri setinde yüksek bellek kullanımı gösterirken, 100k veri setinde bu kullanım düşmekte, 250k veri setinde tekrar artmakta ve 500k veri setinde en düşük seviyeye ulaşmaktadır. Couchbase, 50k veri setinde düşük bellek kullanımı sergilemekte, 100k veri setinde yüksek seviyelere ulaşmakta, 250k veri setinde tekrar düşmekte ve 500k veri setinde yeniden yüksek seviyelere ulaşmaktadır. HBase, 50k veri setinde yüksek bellek kullanımı sergilerken, 100k veri setinde önemli ölçüde düşmekte, 250k veri setinde artmakta ve 500k veri setinde tekrar düşmektedir.

Cassandra, küçük veri setlerinde yüksek iş çıkarma oranları ve işlemci kullanımı sergilemekle birlikte, veri seti boyutu arttıkça performans dalgalanmaları göstermektedir. MongoDB'nin performansı, veri seti boyutlarına göre dalgalanmalar göstermektedir. İş çıkarma oranlarında ve gecikme sürelerinde gözlemlenen bu dalgalanmalar, MongoDB'nin performansının veri seti boyutlarına bağlı olarak değişkenlik gösterdiğini ve belirli veri seti boyutlarında daha iyi optimize edildiğini düşündürmektedir. Couchbase, veri seti boyutlarına göre performansında dalgalanmalar yaşamakta olup, özellikle 100k veri setinde yüksek iş çıkarma oranı ve yüksek gecikme süresi göstermektedir. HBase, 250k ve 500k veri setlerinde yüksek iş çıkarma oranları ve işlemci kullanımı ile öne çıkmakta, 100k veri setinde ise en düşük gecikme süresine sahiptir. Veritabanlarının performansında, veri seti boyutlarına ve iş parçacığı sayılarına göre optimize edilmiş konfigürasyon ayarları ve veri işleme stratejileri önemlidir. İş parçacığı sayısındaki artış, her zaman performansın tam anlamıyla iyileşmesine neden olmamaktadır ve iş çıkarma oranında tam anlamıyla paralellik göstermemektedir. Bu nedenle, her veritabanının performansının optimize edilmesi için sistem ayarları dikkatlice yapılandırılmalıdır.

5.2.2. Test – 2

%95 okuma %5 güncelleme üzerinde oluşturulan test ortamında 250K verisetinde 4, 8 ve 12 iş parçacığı kullanılarak testler gerçekleştirilmiştir. Şekil 5.4'de yapılan testlerden elde edilen sonuçlar görselleştirilmiştir.



Şekil 5.4. Test – 2 Sonucu.

Şekil 5.4(A), farklı iş parçacığı sayılarına göre 250k veri seti boyutunda dört NoSQL veritabanının iş çıkarma oranlarını göstermektedir. Cassandra, 4 iş parçacığında en yüksek iş çıkarma oranına ulaşmakta, 8 ve 12 iş parçacığında ise bu oran önemli ölçüde düşmektedir. MongoDB, 8 iş parçacığında en yüksek iş çıkarma oranını sergilemekte, 4 iş parçacığında biraz daha düşük ve 12 iş parçacığında en düşük oranı göstermektedir. Couchbase, iş parçacığı sayısına bağlı olarak iş çıkarma oranında değişiklik göstermekte, 4 ve 8 iş parçacığında artış sergileyerek 12 iş parçacığında ise düşüş göstermektedir. HBase, 4 ve 8 iş parçacığında benzer yüksek iş çıkarma oranlarına sahipken, 12 iş parçacığında bu oran düşmektedir.

Şekil 5.4(B), farklı iş parçacığı sayılarına göre 250k veri seti boyutunda dört NoSQL veritabanının gecikme sürelerini göstermektedir. Cassandra, 4 iş parçacığında en yüksek gecikme süresine sahipken, 8 iş parçacığında bu süre önemli ölçüde düşmekte ve 12 iş parçacığında ise 8 iş parçacığı ile benzer bir gecikme süresi sergilemektedir. MongoDB, 4 iş parçacığında düşük gecikme süresi sergilerken, 8 iş parçacığında artmakta ve 12 iş parçacığında tekrar düşmektedir. Couchbase, 4 ve 8 iş parçacığında düşük gecikme sürelerine sahip olup, 12 iş parçacığında bu süre önemli ölçüde

artmaktadır. HBase, iş parçacığı sayısı arttıkça gecikme süresinde kademeli bir artış göstermekte ve 12 iş parçacığında en yüksek gecikme süresine ulaşmaktadır.

Şekil 5.4(C), farklı iş parçacığı sayılarına göre 250k veri seti boyutunda dört NoSQL veritabanının işlemci kullanım oranlarını göstermektedir. Cassandra, 4 iş parçacığında yüksek işlemci kullanımı sergilemekte, 8 iş parçacığında belirgin bir düşüş yaşamakta ve 12 iş parçacığında tekrar artmaktadır. MongoDB, 8 iş parçacığında en yüksek işlemci kullanım oranına ulaşmakta, 4 ve 12 iş parçacığında ise bu oran daha düşüktür. Couchbase, 4 iş parçacığında belirgin işlemci kullanımı sergilemekte, 8 iş parçacığında bu oran artmakta ve 12 iş parçacığında tekrar düşmektedir. HBase, 4 iş parçacığında düşük işlemci kullanımı sergilemekte, 8 iş parçacığında düşüş yaşamakta ve 12 iş parçacığında en yüksek seviyeye ulaşmaktadır.

Şekil 5.4(D), farklı iş parçacığı sayılarına göre 250k veri seti boyutunda dört NoSQL veritabanının bellek kullanım oranlarını göstermektedir. Cassandra, 4 ve 12 iş parçacığında yüksek bellek kullanımı sergilemekte, 8 iş parçacığında ise bu oran düşmektedir. MongoDB, 4 iş parçacığında yüksek bellek kullanımı gösterirken, 8 iş parçacığında düşüş yaşanmakta ve 12 iş parçacığında tekrar düşmektedir. Couchbase, 4 iş parçacığında düşük bellek kullanımı gösterirken, 8 ve 12 iş parçacıklarında benzer ve daha yüksek bellek kullanım oranları sergilemektedir. HBase, 4 ve 12 iş parçacığında en yüksek bellek kullanımına ulaşmakta, 8 iş parçacığında ise daha düşük oranlar göstermektedir.

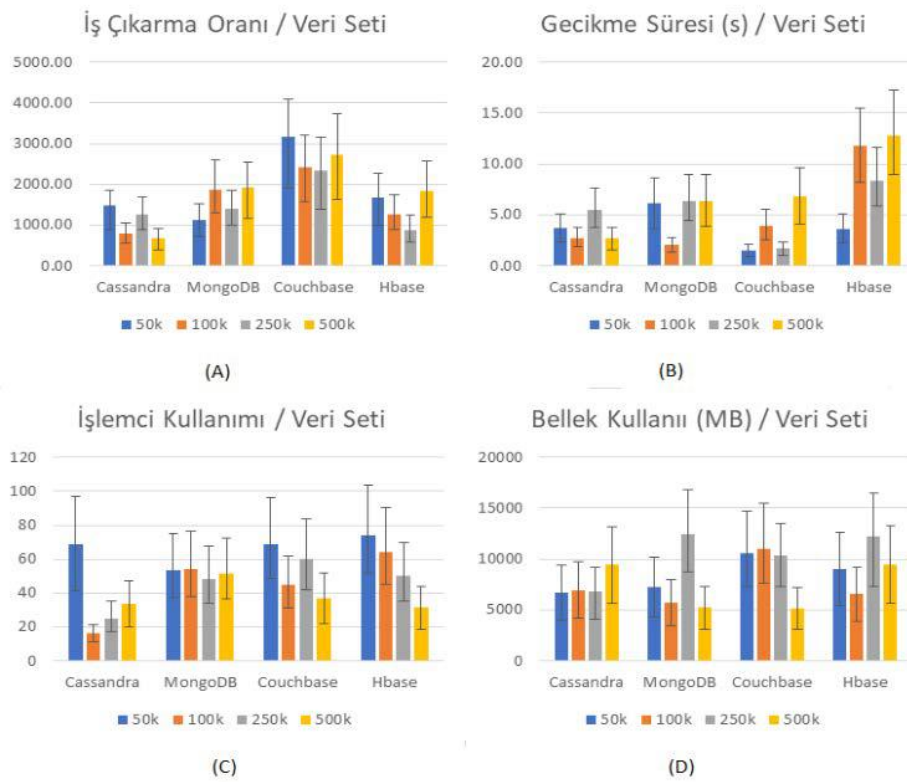
Cassandra, 4 iş parçacığında yüksek iş çıkarma oranı ve işlemci kullanımı sergilemekle birlikte, 8 ve 12 iş parçacığında performans düşüşü göstermektedir. MongoDB, 8 iş parçacığında en yüksek iş çıkarma oranına ve işlemci kullanımına ulaşmakta, ancak 12 iş parçacığında bu oranlar düşmektedir. Couchbase, iş parçacığı sayısına bağlı olarak performansında değişkenlik göstermekte olup, en yüksek performansı 8 iş parçacığında sergilemektedir. 12 iş parçacığında performansı biraz düşerken, 4 iş parçacığında en düşük performansı göstermektedir. Bu durum, Couchbase'in orta seviye iş parçacığı sayısında daha verimli çalıştığını göstermektedir. HBase, 4 ve 8 iş parçacığında yüksek iş çıkarma oranlarına sahipken, 12 iş parçacığında performans düşmektedir. Genel olarak, bellek ve iş parçacığı sayısı arasında belirgin bir ilişki gözlemlenmekle birlikte, bu ilişki her zaman doğrusal değildir. İş parçacığı sayısındaki artış, her zaman performansı tam anlamıyla iyileştirmemektedir ve iş çıkarma oranında

tam anlamıyla paralellik göstermemektedir. Bu nedenle, her veritabanının performansının optimize edilmesi için sistem ayarları dikkatlice yapılandırılmalıdır.

5.3. İş Yüğü C

5.3.1. Test – 1

%100 okuma üzerinde oluşturulan test ortamında 50K, 100K, 250K ve 500K verisetindeki iş çıktı oranı, gecikme süresi, işlemci kullanımı ve bellek kullanımı parametreleriyle testler gerçekleştirilmiştir. Şekil 5.5’de yapılan testlerden elde edilen sonuçlar görselleştirilmiştir.



Şekil 5.5. Test – 1 Sonucu.

Şekil 5.5(A), dört NoSQL veritabanının farklı veri seti boyutlarına göre iş çıkarma oranlarını göstermektedir. Cassandra, 50k veri setinde yüksek bir iş çıkarma oranına sahipken, 100k veri setinde bu oran düşmekte, 250k veri setinde tekrar yükselmekte ve 500k veri setinde tekrar düşmektedir. MongoDB, 50k veri setinde düşük bir iş çıkarma oranına sahipken, 100k veri setinde bu oran artmakta, 250k veri setinde tekrar düşmekte ve 500k veri setinde en yüksek seviyeye ulaşmaktadır. Couchbase, 50k veri setinde en yüksek iş çıkarma oranına sahip olup, 100k veri setinde bu oran düşmekte, 250k veri setinde tekrar artmakta ve 500k veri setinde en yüksek seviyede kalmaktadır.

HBase, 50k veri setinde yüksek bir iş çıkarma oranına sahipken, 100k veri setinde belirgin bir düşüş yaşamakta, 250k veri setinde en düşük seviyeye ulaşmakta ve 500k veri setinde en yüksek seviyeye çıkmaktadır.

Şekil 5.5(B), dört NoSQL veritabanının farklı veri seti boyutlarına göre gecikme sürelerini göstermektedir. Cassandra, 50k veri setinde orta düzeyde bir gecikme süresine sahipken, 100k veri setinde bu süre düşmekte, 250k veri setinde artmakta ve 500k veri setinde tekrar düşmektedir. MongoDB, 50k veri setinde yüksek gecikme süresine sahipken, 100k veri setinde düşmekte, 250k ve 500k veri setlerinde ise yüksek gecikme süreleri sergilemektedir. Couchbase, 50k veri setinde düşük gecikme süresi sergilerken, 100k veri setinde bu süre artmakta, 250k veri setinde tekrar düşmekte ve 500k veri setinde yeniden artmaktadır. HBase, 50k veri setinde düşük gecikme süresi sergilerken, 100k veri setinde artmakta, 250k veri setinde tekrar düşmekte ve 500k veri setinde en yüksek seviyeye ulaşmaktadır.

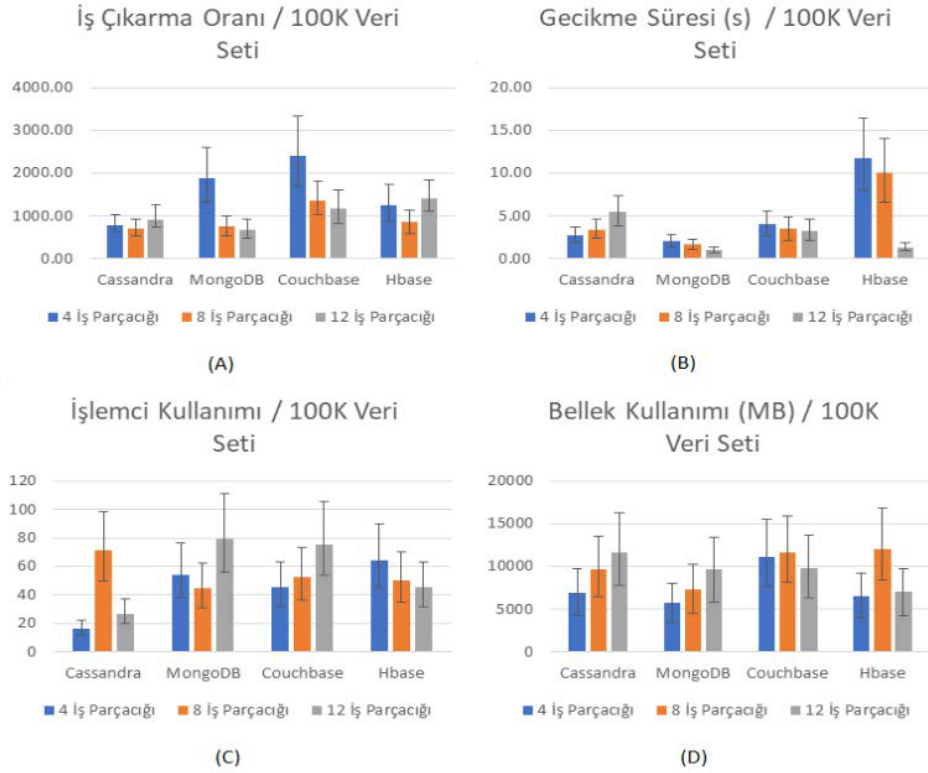
Şekil 5.5(C), dört NoSQL veritabanının farklı veri seti boyutlarına göre işlemci kullanım oranlarını göstermektedir. Cassandra, 50k veri setinde yüksek işlemci kullanımı sergilemekte, 100k veri setinde bu oran önemli ölçüde düşmekte, 250k veri setinde tekrar artmakta ve 500k veri setinde düzenli olarak artmaktadır. MongoDB, 50k ve 100k veri setlerinde yüksek işlemci kullanımı gösterirken, 250k veri setinde bu oran düşmekte ve 500k veri setinde tekrar artmaktadır. Couchbase, 50k veri setinde yüksek işlemci kullanımı sergilemekte, 100k veri setinde bu oran düşmekte, 250k veri setinde artmakta ve 500k veri setinde en düşük seviyeye ulaşmaktadır. HBase, 50k veri setinde yüksek işlemci kullanımı sergilemekte, 100k ve 250k veri setlerinde bu oran düşmekte ve 500k veri setinde daha düşük seviyelere ulaşmaktadır.

Şekil 5.5(D), dört NoSQL veritabanının farklı veri seti boyutlarına göre bellek kullanım oranlarını göstermektedir. Cassandra, 50k veri setinde orta düzeyde bellek kullanımı sergilemekte, 100k veri setinde bu oran artmakta, 250k veri setinde düşmekte ve 500k veri setinde en yüksek seviyeye ulaşmaktadır. MongoDB, 50k veri setinde düşük bellek kullanımı gösterirken, 100k veri setinde düşmekte, 250k veri setinde en yüksek seviyeye ulaşmakta ve 500k veri setinde tekrar düşmektedir. Couchbase, 50k veri setinde orta düzeyde bellek kullanımı sergilemekte, 100k veri setinde artmakta, 250k veri setinde düşmekte ve 500k veri setinde en düşük seviyeye ulaşmaktadır. HBase, 50k veri setinde yüksek bellek kullanımı sergilemekte, 100k veri setinde düşmekte, 250k veri setinde artmakta ve 500k veri setinde tekrar düşmektedir.

Cassandra, küçük veri setlerinde yüksek iş çıkarma oranı ve işlemci kullanımı sergilemekle birlikte, veri seti boyutu arttıkça performans dalgalanmaları göstermektedir. 50k ve 250k veri setlerinde yüksek iş çıkarma oranları gözlemlenirken, 100k ve 500k veri setlerinde performans düşüşü yaşanmaktadır. Bu, Cassandra'nın küçük ve orta büyüklükteki veri setlerinde daha verimli çalıştığını, ancak performansının tutarlı olmadığını göstermektedir. MongoDB, veri seti boyutları arttıkça iş çıkarma oranlarında dalgalanmalar göstermektedir. 50k ve 250k veri setlerinde düşük, 100k ve 500k veri setlerinde ise yüksek iş çıkarma oranlarına sahiptir. Gecikme süreleri ise genel olarak tutarlıdır, ancak 100k veri setinde en düşük gecikme süresi gözlemlenmiştir. Bu durum, MongoDB'nin performansının veri seti boyutlarına göre değişkenlik gösterdiğini ve bazı veri seti boyutlarında optimize edildiğini düşündürmektedir. Couchbase, performansında veri seti boyutlarına göre dalgalanmalar yaşamakta olup, en yüksek iş çıkarma oranını 50k veri setinde göstermektedir. 100k ve 250k veri setlerinde performansı düşerken, 500k veri setinde yeniden artış göstermektedir. Bu, Couchbase'in küçük ve büyük veri setlerinde daha verimli çalıştığını, ancak orta büyüklükteki veri setlerinde performansının düştüğünü göstermektedir. HBase, 250k ve 500k veri setlerinde yüksek iş çıkarma oranları ve işlemci kullanımı ile öne çıkmaktadır. Ancak, 100k veri setinde en düşük gecikme süresine ve 250k veri setinde en düşük iş çıkarma oranına sahiptir. Bu, HBase'in büyük veri setlerinde daha etkili çalıştığını, ancak performansının veri seti boyutlarına göre değişkenlik gösterdiğini göstermektedir. Veritabanlarının performansında, veri seti boyutlarına ve iş parçacığı sayılarına göre optimize edilmiş konfigürasyon ayarları ve veri işleme stratejileri önemlidir. İş parçacığı sayısındaki artış, her zaman performansın iyileşmesine neden olmamaktadır ve iş çıkarma oranında doğrusal bir artış sağlamamaktadır. Bu nedenle, her veritabanının performansının optimize edilmesi için sistem ayarları dikkatlice yapılandırılmalıdır.

5.3.2. Test – 2

%100 okuma üzerinde oluşturulan test ortamında 100K verisetinde 4, 8 ve 12 iş parçacığı kullanılarak testler gerçekleştirilmiştir. Şekil 5.6'de yapılan testlerden elde edilen sonuçlar görselleştirilmiştir.



Şekil 5.6. Test – 2 Sonucu.

Şekil 5.6(A), dört NoSQL veritabanının farklı iş parçacığı sayılarına göre 100k veri seti boyutunda iş çıkarma oranlarını göstermektedir. Cassandra, 12 iş parçacığında en yüksek iş çıkarma oranına sahipken, 4 iş parçacığında bu oran düşmekte ve 8 iş parçacığında en düşük seviyeye inmektedir. MongoDB, 4 iş parçacığında en yüksek iş çıkarma oranına ulaşmakta, 8 ve 12 iş parçacığında ise daha düşük oranlar göstermektedir. Couchbase, 4 iş parçacığında en yüksek iş çıkarma oranına sahip olup, 8 ve 12 iş parçacıklarında bu oran düzenli olarak azalmaktadır. HBase, 12 iş parçacığında en yüksek iş çıkarma oranını sergilemekte, 4 iş parçacığında orta düzeyde ve 8 iş parçacığında ise en düşük oranları göstermektedir.

Şekil 5.6(B), farklı iş parçacığı sayılarına göre 100k veri seti boyutunda dört NoSQL veritabanının gecikme sürelerini göstermektedir. Cassandra, 4, 8 ve 12 iş parçacığında düzenli olarak artan gecikme sürelerine sahiptir. MongoDB, 4 iş parçacığında yüksek bir gecikme süresi sergilerken, 8 ve 12 iş parçacıklarında bu süre azalmaktadır. Couchbase, 4 iş parçacığında yüksek bir gecikme süresine sahipken, 8 ve 12 iş parçacıklarında bu süre azalmaktadır. HBase, 4 ve 8 iş parçacığında yüksek gecikme sürelerine sahipken, 12 iş parçacığında bu süre önemli ölçüde düşmektedir.

Şekil 5.6(C), farklı iş parçacığı sayılarına göre 100k veri seti boyutunda dört NoSQL veritabanının işlemci kullanım oranlarını göstermektedir. Cassandra, 4 iş parçacığında düşük işlemci kullanımı sergilerken, 8 iş parçacığında bu oran en yüksek seviyeye ulaşmakta ve 12 iş parçacığında tekrar düşmektedir. MongoDB, 4 iş parçacığında yüksek işlemci kullanımı gösterirken, 8 iş parçacığında bu oran düşmekte ve 12 iş parçacığında en yüksek seviyeye ulaşmaktadır. Couchbase, 4 iş parçacığında en düşük işlemci kullanımı sergilerken, 8 ve 12 iş parçacığında bu oran artmaktadır. HBase, 4 iş parçacığında yüksek işlemci kullanımı sergilerken, 8 ve 12 iş parçacığında bu oran düşmektedir.

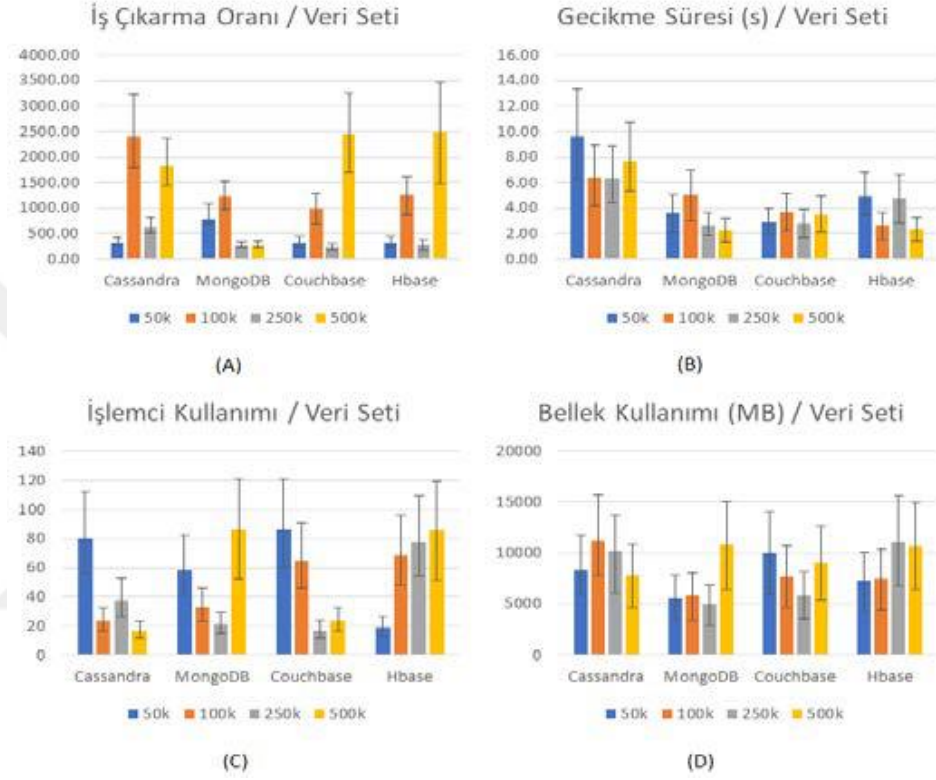
Şekil 5.6(D), farklı iş parçacığı sayılarına göre 100k veri seti boyutunda dört NoSQL veritabanının bellek kullanım oranlarını göstermektedir. Cassandra, 4, 8 ve 12 iş parçacığında düzenli olarak artan bellek kullanımı sergilemektedir. MongoDB, 4 iş parçacığında en düşük bellek kullanımına sahipken, 8 iş parçacığında artmakta ve 12 iş parçacığında en yüksek seviyeye ulaşmaktadır. Couchbase, 4 iş parçacığında yüksek bellek kullanımı sergilemekte, 8 iş parçacığında bu oran artmakta ve 12 iş parçacığında tekrar düşmektedir. HBase, 8 iş parçacığında en yüksek bellek kullanımını sergilemekte olup, 4 iş parçacığında orta seviyede ve 12 iş parçacığında daha düşük seviyelerde kalmaktadır.

Cassandra, 12 iş parçacığında en yüksek iş çıkarma oranına ulaşmakta, ancak gecikme sürelerinde düzenli bir artış göstermektedir. MongoDB, 4 iş parçacığında en yüksek iş çıkarma oranına sahipken, 8 ve 12 iş parçacıklarında daha düşük oranlar göstermekte ve bellek kullanımı düzenli olarak artmaktadır. Couchbase, 4 iş parçacığında en yüksek iş çıkarma oranına sahip olup, 8 ve 12 iş parçacığında performans düşmektedir. HBase, 12 iş parçacığında en yüksek iş çıkarma oranına ve düşük gecikme sürelerine sahip olup, 4 ve 8 iş parçacığında daha düşük performans göstermektedir. Genel olarak, iş parçacığı sayısındaki artışın performans üzerinde her zaman olumlu bir etkisi olmamaktadır ve işlemci ile bellek kullanımı arasında doğrusal olmayan bir ilişki bulunmaktadır. Her veritabanının performansının optimize edilmesi için iş parçacığı sayısı ve diğer sistem ayarlarının dikkatlice yapılandırılması gerekmektedir.

5.4. İş Yüğü D

5.4.1. Test – 1

%75 okuma %25 ekleme üzerinde oluşturulan test ortamında 50K, 100K, 250K ve 500K verisetindeki iş çıktı oranı, gecikme süresi, işlemci kullanımı ve bellek kullanımı parametreleriyle testler gerçekleştirilmiştir. Şekil 5.7’de yapılan testlerden elde edilen sonuçlar görselleştirilmiştir.



Şekil 5.7. Test – 1 Sonucu.

Şekil 5.7(A), dört NoSQL veritabanının farklı veri seti boyutlarına göre iş çıkarma oranlarını göstermektedir. Cassandra, 100k veri setinde en yüksek iş çıkarma oranına sahipken, 250k veri setinde daha düşük ve 50k veri setinde ise en düşük oranları göstermektedir. 500k veri setinde tekrar yüksek bir iş çıkarma oranına ulaşmaktadır. MongoDB, 100k veri setinde yüksek bir iş çıkarma oranı sergilemekte, 50k ve 250k veri setlerinde daha düşük oranlar göstermekte ve 500k veri setinde en düşük oranı sergilemektedir. Couchbase, 500k veri setinde en yüksek iş çıkarma oranına sahipken, 50k ve 250k veri setlerinde daha düşük oranlar göstermekte ve 100k veri setinde tekrar yüksek bir iş çıkarma oranına ulaşmaktadır. HBase, 500k veri setinde en yüksek iş çıkarma oranına sahipken, 100k veri setinde yüksek iş çıkarma oranına sahip olup, 50k ve 250k veri setlerinde daha düşük oranlar göstermektedir.

Şekil 5.7(B), dört NoSQL veritabanının farklı veri seti boyutlarına göre gecikme sürelerini göstermektedir. Cassandra, 50k veri setinde en yüksek gecikme süresine sahipken, 100k ve 250k veri setlerinde daha düşük, 500k veri setinde ise tekrar yüksek gecikme süreleri sergilemektedir. MongoDB, 50k veri setinde orta düzeyde gecikme süresine sahipken, 100k veri setinde en yüksek gecikme süresini göstermekte, 250k veri setinde bu süre düşmekte ve 500k veri setinde en düşük seviyeye inmektedir. Couchbase, 50k veri setinde düşük bir gecikme süresine sahip olup, 100k veri setinde bu süre artmakta, 250k veri setinde düşmekte ve 500k veri setinde tekrar artmaktadır. HBase, 50k veri setinde yüksek gecikme süresine sahipken, 100k veri setinde daha düşük bir gecikme süresi göstermekte, 250k veri setinde tekrar artmakta ve 500k veri setinde en düşük seviyeye inmektedir.

Şekil 5.7(C), dört NoSQL veritabanının farklı veri seti boyutlarına göre işlemci kullanım oranlarını göstermektedir. Cassandra, 50k veri setinde en yüksek işlemci kullanım oranına sahipken, 100k veri setinde bu oran düşmekte, 250k veri setinde tekrar artmakta ve 500k veri setinde tekrar düşmektedir. MongoDB, 50k veri setinde yüksek işlemci kullanımı sergilemekte, 100k veri setinde bu oran düşmekte, 250k veri setinde en düşük seviyeye ulaşmakta ve 500k veri setinde en yüksek seviyeye çıkmaktadır. Couchbase, 50k veri setinde en yüksek işlemci kullanım oranına sahipken, 100k veri setinde bu oran düşmekte, 250k veri setinde en düşük seviyeye ulaşmakta ve 500k veri setinde tekrar artmaktadır. HBase, 50k veri setinden sonra düzenli olarak artan işlemci kullanım oranlarına sahiptir.

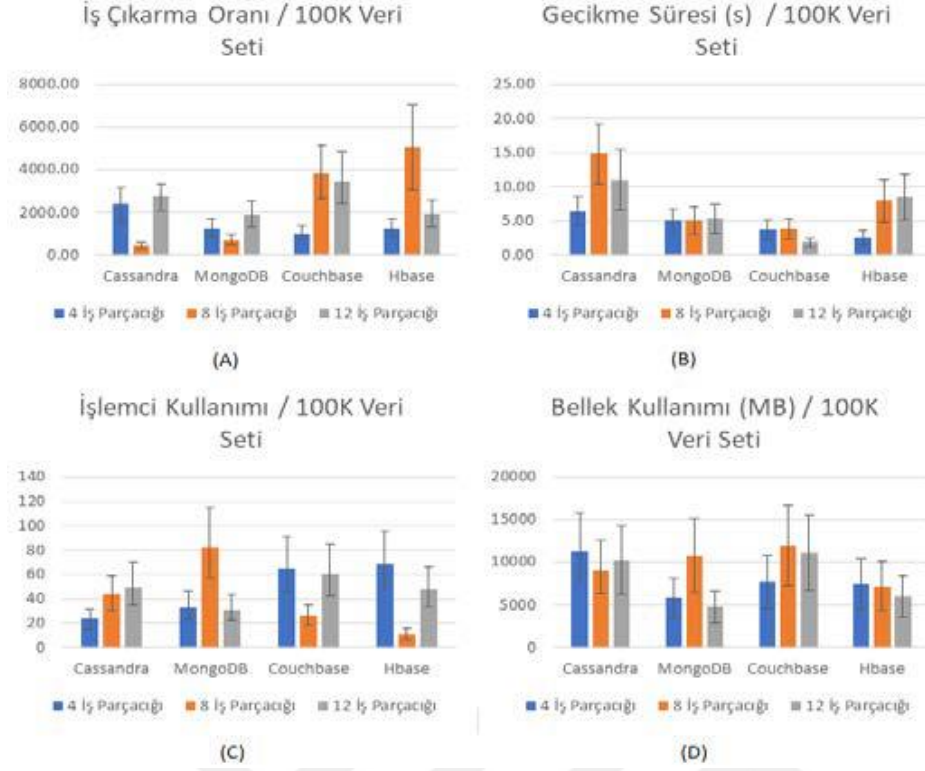
Şekil 5.7(D), dört NoSQL veritabanının farklı veri seti boyutlarına göre bellek kullanım oranlarını göstermektedir. Cassandra, 50k veri setinde yüksek bellek kullanımı sergilemekte, 100k veri setinde bu oran artmakta, 250k veri setinde tekrar düşmekte ve 500k veri setinde en düşük seviyeye ulaşmaktadır. MongoDB, 50k ve 100k veri setlerinde benzer bellek kullanımı göstermekte, 250k veri setinde en düşük seviyeye ulaşmakta ve 500k veri setinde en yüksek seviyeye çıkmaktadır. Couchbase, 50k veri setinde yüksek bellek kullanımı sergilemekte, 100k ve 250k veri setlerinde düşmekte ve 500k veri setinde tekrar artmaktadır. HBase, 50k veri setinde düşük bellek kullanımı sergilemekte, 100k veri setinde bu oran artmakta, 250k veri setinde en yüksek seviyeye ulaşmakta ve 500k veri setinde bir miktar düşmektedir.

Cassandra, 100k veri setinde en yüksek iş çıkarma oranına sahipken, 50k ve 250k veri setlerinde daha düşük performans göstermekte ve 500k veri setinde tekrar yüksek

performansa ulaşmaktadır. Bu durum, Cassandra'nın belirli veri seti boyutlarında optimize edilmiş performans gösterdiğini ancak performansının tutarsız olduğunu ortaya koymaktadır. MongoDB, 100k veri setinde yüksek iş çıkarma oranı sergilerken, 500k veri setinde en düşük oranı göstermektedir. Gecikme süreleri açısından ise genellikle tutarlı ve düşük gecikme süreleri göstermektedir. Bu, MongoDB'nin küçük ve orta büyüklükteki veri setlerinde daha verimli çalıştığını göstermektedir. Couchbase, 500k veri setinde en yüksek iş çıkarma oranına sahip olup, veri seti boyutları arasında dalgalanmalar yaşamaktadır. Bu durum, Couchbase'in büyük veri setlerinde daha iyi performans gösterdiğini ancak performansının veri seti boyutlarına göre tutarsız olduğunu göstermektedir. HBase, 500k veri setinde en yüksek iş çıkarma oranına sahip olup, diğer veri seti boyutlarında daha düşük oranlar göstermektedir. Gecikme süreleri açısından, HBase genellikle düşük gecikme süreleri sergilemekte ve büyük veri setlerinde daha iyi performans göstermektedir. Her veritabanının performansının optimize edilmesi için sistem ayarları dikkatlice yapılandırılmalıdır.

5.4.2. Test – 2

%75 okuma %25 ekleme üzerinde oluşturulan test ortamında 100K verisetinde 4, 8 ve 12 iş parçacığı kullanılarak testler gerçekleştirilmiştir. Şekil 5.8'de yapılan testlerden elde edilen sonuçlar görselleştirilmiştir.



Şekil 5.8. Test – 2 Sonucu.

Şekil 5.8(A), farklı iş parçacığı sayılarına göre 100k veri seti boyutunda dört NoSQL veritabanının iş çıkarma oranlarını göstermektedir. Cassandra, 4 iş parçacığında yüksek bir iş çıkarma oranına sahipken, 8 iş parçacığında bu oran önemli ölçüde düşmekte ve 12 iş parçacığında tekrar artmaktadır. MongoDB, 4 iş parçacığında yüksek bir iş çıkarma oranı sergilemekte, 8 iş parçacığında bu oran düşmekte ve 12 iş parçacığında tekrar artmaktadır. Couchbase, 4 iş parçacığında orta düzeyde bir iş çıkarma oranına sahipken, 8 iş parçacığında bu oran artmakta ve 12 iş parçacığında bir miktar düşmektedir. HBase, 8 iş parçacığında en yüksek iş çıkarma oranını göstermekte olup, 4 ve 12 iş parçacıklarında bu oran daha düşük seviyelerde kalmaktadır.

Şekil 5.8(B), farklı iş parçacığı sayılarına göre 100k veri seti boyutunda dört NoSQL veritabanının gecikme sürelerini göstermektedir. Cassandra, 4 iş parçacığında düşük bir gecikme süresine sahipken, 8 iş parçacığında bu süre önemli ölçüde artmakta ve 12 iş parçacığında düşmektedir. MongoDB, 4 iş parçacığında nispeten düşük bir gecikme süresi gösterirken, 8 ve 12 iş parçacıklarında gecikme süreleri benzer ve daha yüksek seviyelerde seyretmektedir. Couchbase, 4 iş parçacığında düşük bir gecikme süresine sahip olup, 8 iş parçacığında bu süre artmakta ve 12 iş parçacığında tekrar

düşmektedir. HBase, 4 iş parçacığında nispeten düşük bir gecikme süresi sergilerken, 8 iş parçacığında bu süre önemli ölçüde artmakta ve 12 iş parçacığında tekrar artmaktadır.

Şekil 5.8(C), farklı iş parçacığı sayılarına göre 100k veri seti boyutunda dört NoSQL veritabanının işlemci kullanım oranlarını göstermektedir. Cassandra, 4 iş parçacığında düşük işlemci kullanımı sergilemekte, 8 iş parçacığında bu oran önemli ölçüde artmakta ve 12 iş parçacığında en yüksek seviyeye ulaşmaktadır. MongoDB, 4 iş parçacığında orta düzeyde işlemci kullanımı sergilerken, 8 iş parçacığında en yüksek seviyeye ulaşmakta ve 12 iş parçacığında tekrar düşmektedir. Couchbase, 4 iş parçacığında yüksek işlemci kullanımı göstermekte, 8 iş parçacığında bu oran düşmekte ve 12 iş parçacığında tekrar artmaktadır. HBase, 4 iş parçacığında yüksek işlemci kullanımı sergilemekte, 8 iş parçacığında bu oran düşmekte ve 12 iş parçacığında tekrar artmaktadır.

Şekil 5.8(D), farklı iş parçacığı sayılarına göre 100k veri seti boyutunda dört NoSQL veritabanının bellek kullanım oranlarını göstermektedir. Cassandra, 4 iş parçacığında yüksek bellek kullanımı sergilemekte, 8 iş parçacığında bu oran düşmekte ve 12 iş parçacığında tekrar artmaktadır. MongoDB, 4 iş parçacığında düşük bellek kullanımı sergilemekte, 8 iş parçacığında bu oran önemli ölçüde artmakta ve 12 iş parçacığında tekrar düşmektedir. Couchbase, 4 iş parçacığında düşük bellek kullanımı göstermekte, 8 iş parçacığında bu oran artmakta ve 12 iş parçacığında tekrar düşmektedir. HBase, 4 iş parçacığında yüksek bellek kullanımı sergilemekte, 8 iş parçacığında bu oran düşmekte ve 12 iş parçacığında en düşük seviyeye inmektedir. Cassandra, 4 iş parçacığında yüksek iş çıkarma oranına sahipken, 8 iş parçacığında performansı düşmekte ve 12 iş parçacığında tekrar artmaktadır. Gecikme süreleri ise iş parçacığı sayısına göre önemli bir değişiklik göstermemektedir. Bu durum, Cassandra'nın iş parçacığı sayısındaki artışa karşı performansının dalgalandığını göstermektedir. MongoDB, iş çıkarma oranlarında 4 iş parçacığında yüksek performans sergilemekte, ancak 8 ve 12 iş parçacıklarında bu oranlar düşmektedir. Gecikme süreleri ise 4, 8 ve 12 iş parçacıklarında neredeyse aynı kalmaktadır. Bu, MongoDB'nin düşük iş parçacığı sayısında daha verimli çalıştığını göstermektedir.

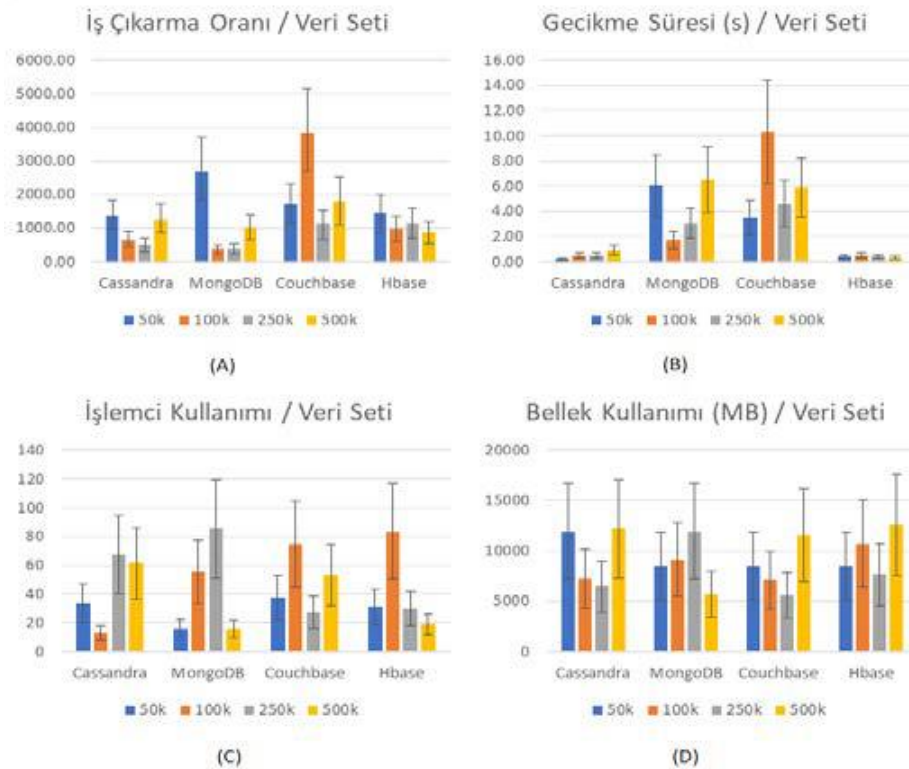
Couchbase, 4 iş parçacığında en düşük iş çıkarma oranını göstermekte, 8 iş parçacığında bu oran artmakta ve 12 iş parçacığında tekrar düşmektedir. Gecikme süreleri açısından ise, 12 iş parçacığında en düşük gecikme süresine sahiptir. Bu,

Couchbase'in yüksek iş parçacığı sayısında daha verimli çalıştığını, ancak düşük iş parçacığı sayısında performansının düştüğünü göstermektedir. HBase, 8 iş parçacığında en yüksek iş çıkarma oranını sergilemekte, ancak işlemci kullanımı en düşük seviyededir. 4 ve 12 iş parçacıklarında ise performansı daha düşük seviyelerde kalmaktadır. Bu durum, HBase'in iş parçacığı sayısına karşı performansının tutarsız olduğunu göstermektedir. Genel olarak, bellek ve iş parçacığı sayısı arasında belirgin ve doğrusal bir ilişki gözlemlenmemektedir. İş parçacığı sayısındaki artış her zaman performansı iyileştirmemekte ve iş çıkarma oranında doğrusal bir artış sağlamamaktadır. Bu nedenle, her veritabanının performansının optimize edilmesi için iş parçacığı sayısı ve diğer sistem ayarlarının dikkatlice yapılandırılması gerekmektedir.

5.5. İş Yüğü E

5.5.1. Test – 1

%75 ekleme %25 güncelleme üzerinde oluşturulan test ortamında 50K, 100K, 250K ve 500K verisetindeki iş çıktı oranı, gecikme süresi, işlemci kullanımı ve bellek kullanımı parametreleriyle testler gerçekleştirilmiştir. Şekil 5.9'de yapılan testlerden elde edilen sonuçlar görselleştirilmiştir.



Şekil 5.9. Test – 1 Sonucu.

Şekil 5.9(A), dört NoSQL veritabanının farklı veri seti boyutlarına göre iş çıkarma oranlarını göstermektedir. Cassandra, 50k veri setinde yüksek iş çıkarma oranına sahipken, 100k veri setinde bu oran düşmekte, 250k veri setinde tekrar artmakta ve 500k veri setinde en düşük seviyeye inmektedir. MongoDB, 50k veri setinde yüksek iş çıkarma oranı sergilemekte, 100k veri setinde bu oran düşmekte, 250k veri setinde tekrar artmakta ve 500k veri setinde de artmaktadır. Couchbase, 100k veri setinde en yüksek iş çıkarma oranına sahipken, 50k ve 250k veri setlerinde daha düşük oranlar göstermekte ve 500k veri setinde tekrar yüksek bir iş çıkarma oranına ulaşmaktadır. HBase ise 50k veri setinde en yüksek iş çıkarma oranına sahipken, 100k veri setinde bu oran düşmekte, 250k veri setinde tekrar artmakta ve 500k veri setinde en düşük seviyeye inmektedir.

Şekil 5.9(B), dört NoSQL veritabanının farklı veri seti boyutlarına göre gecikme sürelerini göstermektedir. Cassandra, 50k veri setinde en düşük gecikme süresine sahip olup, 100k veri setinde bu süre artmakta, 250k veri setinde tekrar düşmekte ve 500k veri setinde en yüksek seviyeye ulaşmaktadır. MongoDB, 50k veri setinde yüksek bir gecikme süresine sahipken, 100k veri setinde bu süre düşmekte, 250k veri setinde tekrar artmakta ve 500k veri setinde en yüksek seviyeye ulaşmaktadır. Couchbase, 50k veri setinde düşük bir gecikme süresine sahip olup, 100k veri setinde bu süre önemli ölçüde artmakta, 250k veri setinde tekrar düşmekte ve 500k veri setinde tekrar artmaktadır. HBase, 50k ve 500k veri setlerinde oldukça düşük gecikme süreleri sergilerken, 100k veri setinde en yüksek gecikme süresine sahip olup, 250k verisetinde ise bu süre tekrar düşmektedir.

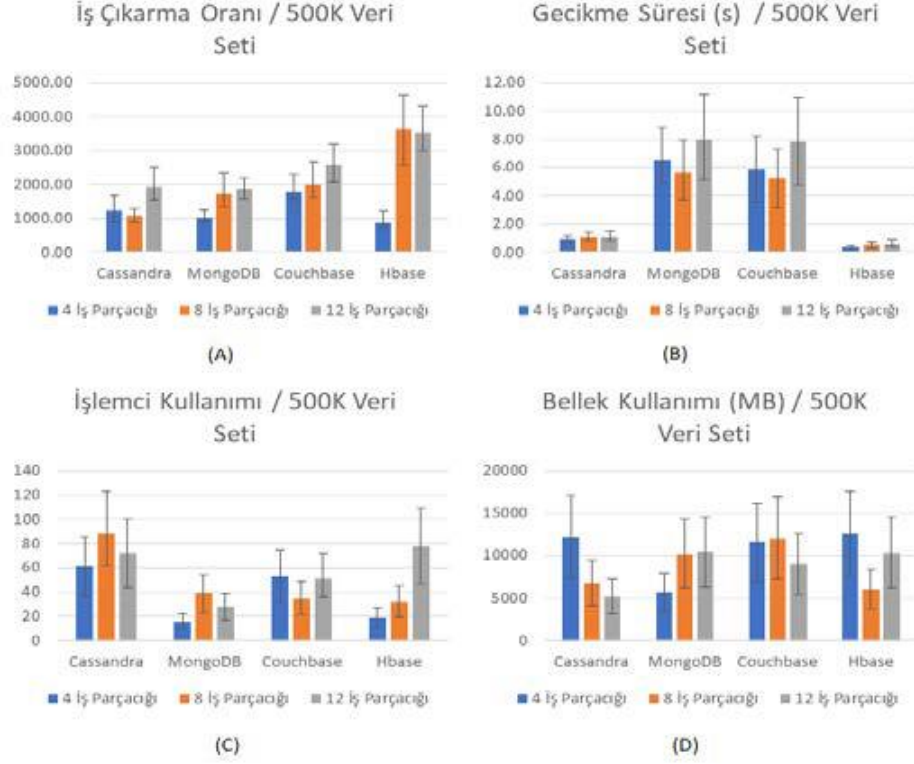
Şekil 5.9(C), dört NoSQL veritabanının farklı veri seti boyutlarına göre işlemci kullanım oranlarını göstermektedir. Cassandra, 50k veri setinde orta düzeyde işlemci kullanımı sergilerken, 100k veri setinde bu oran önemli ölçüde düşmekte, 250k veri setinde tekrar artmakta ve 500k veri setinde tekrar düşmektedir. MongoDB, 50k veri setinde düşük işlemci kullanımı göstermekte, 100k ve 250k veri setlerinde bu oran artmakta ve 500k veri setinde tekrar düşmektedir. Couchbase, 50k veri setinde orta düzeyde işlemci kullanımı sergilemekte, 100k veri setinde bu oran artmakta, 250k veri setinde düşmekte ve 500k veri setinde tekrar artmaktadır. HBase, 50k veri setinde yüksek işlemci kullanımı sergilerken, 100k veri setinde bu oran daha da artmakta, 250k veri setinde düşmekte ve 500k veri setinde tekrar düşmektedir.

Şekil 5.9(D), dört NoSQL veritabanının farklı veri seti boyutlarına göre bellek kullanım oranlarını göstermektedir. Cassandra, 50k veri setinde yüksek bellek kullanımı sergilemekte, 100k veri setinde bu oran düşmekte, 250k veri setinde daha da düşmekte ve 500k veri setinde en yüksek seviyeye ulaşmaktadır. MongoDB, 50k veri setinde yüksek bellek kullanımı göstermekte, 100k veri setinde bu oran artmakta, 250k veri setinde en yüksek seviyeye ulaşmakta ve 500k veri setinde tekrar düşmektedir. Couchbase, 50k veri setinde yüksek bellek kullanımı sergilemekte, 100k veri setinde bu oran artmakta, 250k veri setinde düşmekte ve 500k veri setinde tekrar artmaktadır. HBase, 50k veri setinde yüksek bellek kullanımı sergilerken, 100k veri setinde bu oran artmakta, 250k veri setinde düşmekte ve 500k veri setinde en yüksek seviyeye ulaşmaktadır.

Cassandra, 50k veri setinde yüksek iş çıkarma oranına ve düşük gecikme süresine sahipken, 100k ve 250k veri setlerinde performansı düşmekte, 500k veri setinde ise tekrar artmaktadır. Bu, Cassandra'nın performansının veri seti boyutuna göre dalgalandığını göstermektedir. MongoDB, 50k veri setinde nispeten yüksek iş çıkarma oranı sergilemekte ancak gecikme süresi yüksek kalmaktadır. 100k veri setinde en düşük iş çıkarma oranına ve en düşük gecikme süresine sahip olan MongoDB, 250k ve 500k veri setlerinde performansında belirgin dalgalanmalar göstermektedir. Özellikle 500k veri setinde iş çıkarma oranı tekrar artmakta ancak gecikme süresi yüksek kalmaktadır. Couchbase, veri seti boyutlarına göre performansında dalgalanmalar göstermektedir. 100k veri setinde en yüksek iş çıkarma oranını sergileyen Couchbase, 50k ve 250k veri setlerinde daha düşük performans göstermekte, 500k veri setinde ise tekrar artmaktadır. Bu, Couchbase'in veri seti boyutuna bağlı olarak tutarsız performans gösterdiğini ortaya koymaktadır. HBase, 250k veri setinde en yüksek iş çıkarma oranına ve düşük gecikme süresine sahip olup, 100k veri setinde performansı düşmekte, 250k veri setinde tekrar artmakta ve 500k veri setinde yeniden düşmektedir. Bu, HBase'in performansının da veri seti boyutuna göre dalgalandığını göstermektedir. Veritabanlarının performansının optimize edilmesi için, iş parçacığı sayısı ve diğer sistem ayarlarının dikkatlice yapılandırılması gerekmektedir. İş parçacığı sayısındaki artış her zaman performansın iyileşmesine neden olmamakta ve iş çıkarma oranında doğrusal bir artış sağlamamaktadır.

5.5.2. Test – 2

%75 ekleme %25 güncelleme üzerinde oluşturulan test ortamında 500K verisetinde 4, 8 ve 12 iş parçacığı kullanılarak testler gerçekleştirilmiştir. Şekil 5.10'de yapılan testlerden elde edilen sonuçlar görselleştirilmiştir.



Şekil 5.10. Test – 2 Sonucu.

Şekil 5.10(A), farklı iş parçacığı sayıları altında dört NoSQL veritabanının 500k veri seti boyutlarına göre iş çıkarma oranlarını göstermektedir. Cassandra, 4 iş parçacığında orta düzeyde bir iş çıkarma oranına sahipken, 8 iş parçacığında bu oran biraz düşmekte ve 12 iş parçacığında önemli ölçüde artmaktadır. MongoDB, 4 iş parçacığında düşük bir iş çıkarma oranı sergilemekte, 8 iş parçacığında bu oran artmakta ve 12 iş parçacığında en yüksek seviyeye ulaşmaktadır. Couchbase, 4 iş parçacığında orta düzeyde bir iş çıkarma oranına sahip olup, 8 iş parçacığında bu oran artmakta ve 12 iş parçacığında en yüksek seviyeye çıkmaktadır. HBase, 4 iş parçacığında düşük bir iş çıkarma oranı sergilerken, 8 iş parçacığında bu oran önemli ölçüde artmakta ve 12 iş parçacığında bu yüksek seviyeyi korumaktadır.

Şekil 5.10(B), farklı iş parçacığı sayıları altında dört NoSQL veritabanının 500k veri seti boyutlarına göre gecikme sürelerini göstermektedir. Cassandra, 4, 8 ve 12 iş parçacığında oldukça düşük ve benzer gecikme süreleri sergilemektedir. MongoDB, 4

iş parçacığında yüksek bir gecikme süresine sahipken, 8 iş parçacığında bu süre biraz düşmekte ve 12 iş parçacığında tekrar artmaktadır. Couchbase, 4 iş parçacığında orta düzeyde bir gecikme süresi sergilemekte, 8 iş parçacığında bu süre düşmekte ve 12 iş parçacığında en yüksek seviyeye ulaşmaktadır. HBase ise 4 iş parçacığında düşük bir gecikme süresi sergilemekte, 8 iş parçacığında bu süre artmakta ve 12 iş parçacığında daha da artarak en yüksek seviyeye ulaşmaktadır.

Şekil 5.10(C), farklı iş parçacığı sayıları altında dört NoSQL veritabanının 500k veri seti boyutlarına göre işlemci kullanım oranlarını göstermektedir. Cassandra, 4 iş parçacığında yüksek bir işlemci kullanımı sergilemekte, 8 iş parçacığında bu oran artmakta ve 12 iş parçacığında tekrar düşmektedir. MongoDB, 4 iş parçacığında düşük bir işlemci kullanımı gösterirken, 8 iş parçacığında bu oran artmakta ve 12 iş parçacığında tekrar düşmektedir. Couchbase, 4 iş parçacığında orta düzeyde bir işlemci kullanımı sergilemekte, 8 iş parçacığında bu oran düşmekte ve 12 iş parçacığında tekrar artmaktadır. HBase, 4 iş parçacığında düşük bir işlemci kullanımı sergilemekte, 8 iş parçacığında bu oran artmakta ve 12 iş parçacığında en yüksek seviyeye ulaşmaktadır.

Şekil 5.10(D), dört NoSQL veritabanının (Cassandra, MongoDB, Couchbase ve HBase) farklı iş parçacığı sayıları altında 500k veri seti boyutlarına göre bellek kullanım oranlarını göstermektedir. Cassandra, 4 iş parçacığında en yüksek bellek kullanımına sahipken, 8 iş parçacığında bu oran düşmekte ve 12 iş parçacığında tekrar artmaktadır. MongoDB, 4 iş parçacığında düşük bellek kullanımı sergilemekte, 8 iş parçacığında bu oran artmakta ve 12 iş parçacığında daha da artmaktadır. Couchbase, 4 iş parçacığında düşük bellek kullanımı göstermekte, 8 iş parçacığında bu oran artmakta ve 12 iş parçacığında daha da artmaktadır. HBase ise 4 iş parçacığında yüksek bellek kullanımı sergilemekte, 8 iş parçacığında bu oran önemli ölçüde düşmekte ve 12 iş parçacığında tekrar artmaktadır.

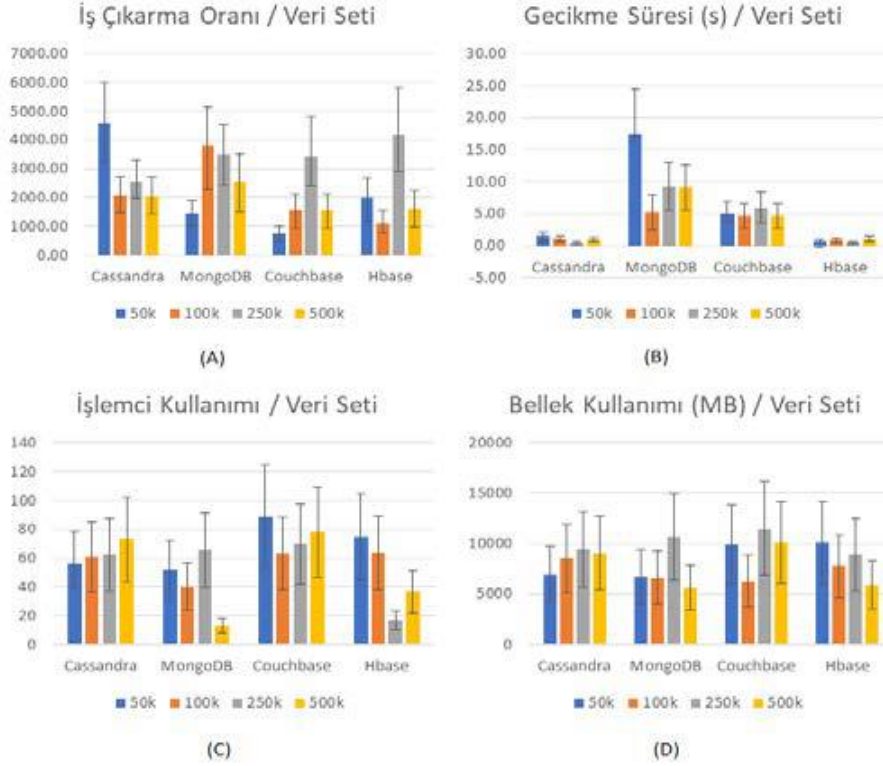
Cassandra, 4 iş parçacığında orta düzeyde iş çıkarma oranına ve düşük gecikme süresine sahipken, 8 iş parçacığında performansı düşmekte ve 12 iş parçacığında iş çıkarma oranı artmaktadır. Bu, Cassandra'nın iş parçacığı sayısındaki artışa karşı performansının dalgalandığını göstermektedir. MongoDB, 4 iş parçacığında düşük iş çıkarma oranı ve yüksek gecikme süresi sergilemekte, 8 iş parçacığında performansı artmakta ve 12 iş parçacığında en yüksek iş çıkarma oranına ulaşmaktadır. Bu durum, MongoDB'nin yüksek iş parçacığı sayısında daha verimli çalıştığını göstermektedir.

Couchbase, 4 iş parçacığında en düşük iş çıkarma oranını ve orta düzeyde bellek kullanımı sergilemekte, 8 iş parçacığında iş çıkarma oranı artmakta ve 12 iş parçacığında en yüksek iş çıkarma oranına ve en düşük bellek kullanımına ulaşmaktadır. Bu, Couchbase'in iş parçacığı sayısındaki artışa karşı performansının iyileştiğini ancak bellek kullanımının azaldığını göstermektedir. HBase, 4 iş parçacığında düşük iş çıkarma oranı ve gecikme süresi sergilemekte, 8 iş parçacığında performansı önemli ölçüde artmakta ve 12 iş parçacığında yüksek seviyeyi korumaktadır. Bu, HBase'in iş parçacığı sayısındaki artışa karşı performansının iyileştiğini göstermektedir. Genel olarak, bellek ve iş parçacığı sayısı arasında doğrudan bir ilişki gözlemlenmemektedir. Veritabanlarının performansı, iş parçacığı sayısındaki artışla her zaman doğrusal bir şekilde iyileşmemekte ve iş çıkarma oranlarında tam bir paralellik göstermemektedir.

5.6. İş Yüğü F

5.6.1. Test – 1

%50 ekleme %50 okuma-güncelleme-yazma güncelleme üzerinde oluşturulan test ortamında 50K, 100K, 250K ve 500K verisetindeki iş çıktı oranı, gecikme süresi, işlemci kullanımı ve bellek kullanımı parametreleriyle testler gerçekleştirilmiştir. Şekil 5.11'de yapılan testlerden elde edilen sonuçlar görselleştirilmiştir.



Şekil 5.11. Test – 1 Sonucu.

Şekil 5.11(A), dört NoSQL veritabanı sisteminin (Cassandra, MongoDB, Couchbase ve HBase) farklı veri seti boyutlarına (50k, 100k, 250k ve 500k) göre iş çıkarma oranlarını göstermektedir. Cassandra, 50k veri setinde en yüksek iş çıkarma oranına sahipken, 100k ve 250k veri setlerinde bu oran düşmekte ve 500k veri setinde yeniden artmaktadır. MongoDB, 100k veri setinde en yüksek iş çıkarma oranını göstermekte, 50k veri setinde bu oran düşmekte, 250k veri setinde 50k'ya göre daha yüksek, ancak 100k'ya göre daha düşük bir oran sergilemekte ve 500k veri setinde en düşük seviyeye inmektedir. Couchbase, 250k veri setinde en yüksek iş çıkarma oranına sahipken, 50k ve 500k veri setlerinde daha düşük oranlar göstermektedir. 100k veri setinde ise orta düzeyde bir iş çıkarma oranına sahip olup, 50k veri setine göre daha yüksek, ancak 250k veri setine göre daha düşük performans sergilemektedir. HBase, 250k veri setinde en yüksek iş çıkarma oranını sergilemekte, 50k ve 500k veri setlerinde ise daha düşük oranlara sahip olmaktadır.

Şekil 5.11(B), dört NoSQL veritabanı sisteminin (Cassandra, MongoDB, Couchbase ve HBase) farklı veri seti boyutlarına (50k, 100k, 250k ve 500k) göre gecikme sürelerini göstermektedir. Cassandra, 50k veri setinde en yüksek gecikme süresine sahip olup, 100k veri setinde bu süre düşmekte, 250k veri setinde en düşük seviyeye

ulaşmakta ve 500k veri setinde tekrar artmaktadır. MongoDB, 50k veri setinde en yüksek gecikme süresine sahipken, 100k veri setinde bu süre düşmekte, 250k veri setinde artmakta ve 500k veri setinde tekrar düşmektedir. Couchbase, 50k veri setinde düşük bir gecikme süresine sahip olup, 100k veri setinde bu süre düşmekte, 250k veri setinde artmakta ve 500k veri setinde yeniden düşmektedir. HBase ise tüm veri seti boyutlarında oldukça düşük gecikme süreleri sergilemekte olup, en düşük gecikme süresine 250k veri setinde ulaşmakta, 50k veri setinde biraz artmakta, 100k veri setinde daha da artmakta ve 500k veri setinde en yüksek seviyeye ulaşmaktadır.

Şekil 5.11(C), dört NoSQL veritabanı sisteminin (Cassandra, MongoDB, Couchbase ve HBase) farklı veri seti boyutlarına göre işlemci kullanım oranlarını göstermektedir. Cassandra, 50k veri setinde düşük işlemci kullanımı sergilemekte olup, 100k, 250k ve 500k veri setlerinde bu oran artarak en yüksek seviyeye ulaşmaktadır. MongoDB, 50k veri setinde nispeten düşük işlemci kullanımı göstermekte, 100k veri setinde bu oran düşmekte, 250k veri setinde en yüksek seviyeye ulaşmakta ve 500k veri setinde en düşük seviyeye inmektedir. Couchbase, 50k veri setinde yüksek işlemci kullanımı sergilemekte, 100k veri setinde bu oran düşmekte ve ardından 250k ve 500k veri setlerinde düzenli olarak artmaktadır. HBase ise 50k veri setinde yüksek işlemci kullanımı sergilemekte, 100k veri setinde düşmekte, 250k veri setinde en düşük seviyeye inmekte ve 500k veri setinde tekrar artmaktadır.

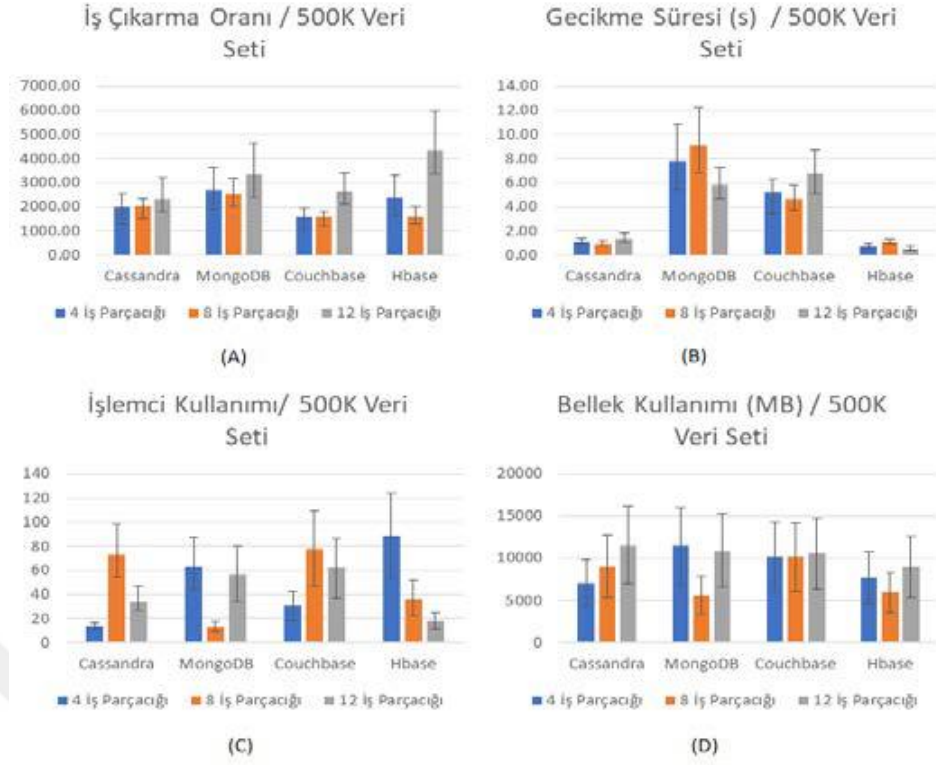
Şekil 5.11(D), dört NoSQL veritabanı sisteminin (Cassandra, MongoDB, Couchbase ve HBase) farklı veri seti boyutlarına göre bellek kullanım oranlarını göstermektedir. Cassandra, 50k veri setinde yüksek bellek kullanımı sergilemekte olup, 100k veri setinde bu oran artmakta, 250k veri setinde de artmaya devam etmekte ve 500k veri setinde en yüksek seviyeye ulaşmaktadır. MongoDB, 50k veri setinde nispeten orta düzeyde bellek kullanımı göstermekte, 100k veri setinde bu oran düşmekte, 250k veri setinde en yüksek seviyeye ulaşmakta ve 500k veri setinde tekrar düşmektedir. Couchbase, 50k veri setinde yüksek bellek kullanımı sergilemekte, 100k veri setinde bu oran düşmekte, 250k veri setinde en yüksek seviyeye ulaşmakta ve 500k veri setinde düşmektedir. HBase ise 50k veri setinde yüksek bellek kullanımı sergilemekte, 100k veri setinde bu oran düşmekte, 250k veri setinde artmakta ve 500k veri setinde tekrar düşmektedir.

Cassandra, 50k veri setinde yüksek iş çıkarma oranına ve en düşük bellek kullanımına sahip olup, gecikme süresi de en yüksek seviyededir. 100k ve 250k veri setlerinde

performansı dalgalanmakta, 500k veri setinde ise iş çıkarma oranı düşmektedir. Bu, Cassandra'nın performansının veri seti boyutuna göre dalgalandığını ve büyük veri setlerinde performansının düştüğünü göstermektedir. MongoDB, 100k veri setinde en yüksek iş çıkarma oranına sahip olup, düşük işlemci kullanımı ve düşük gecikme süresi sergilemektedir. 250k veri setinde işlemci ve bellek kullanımı en yüksek seviyeye ulaşmakta, 500k veri setinde ise en düşük seviyeye inmektedir. Bu, MongoDB'nin performansının veri seti boyutuna göre değişkenlik gösterdiğini ve belirli veri seti boyutlarında daha verimli çalıştığını ortaya koymaktadır. Couchbase, 250k veri setinde en yüksek iş çıkarma oranına sahip olup, 50k ve 250k veri setlerinde bellek kullanımı yüksektir. 100k veri setinde performansı düşmekte ve 500k veri setinde ise iş çıkarma oranı tekrar düşmektedir. Bu durum, Couchbase'in büyük veri setlerinde performansının düştüğünü göstermektedir. HBase, 250k veri setinde en yüksek iş çıkarma oranını ve en düşük gecikme süresini sergilemekte olup, 50k ve 500k veri setlerinde performansı daha düşük seviyelerdedir. Bu, HBase'in belirli veri seti boyutlarında optimize edilmiş performans gösterdiğini ancak genel performansının tutarsız olduğunu göstermektedir. Performans, veri seti boyutuna ve sistem ayarlarına bağlı olarak değişiklik göstermektedir. Bu nedenle, her veritabanının performansının optimize edilmesi için veri seti boyutları ve diğer sistem ayarlarının dikkatlice yapılandırılması gerekmektedir.

5.6.2. Test – 2

%50 ekleme %50 okuma-güncelleme-yazma üzerinde oluşturulan test ortamında 500K verisetinde 4, 8 ve 12 iş parçacığı kullanılarak testler gerçekleştirilmiştir. Şekil 5.12'de yapılan testlerden elde edilen sonuçlar görselleştirilmiştir.



Şekil 5.12. Test – 2 Sonucu.

Şekil 5.12(A), dört NoSQL veritabanı sisteminin (Cassandra, MongoDB, Couchbase ve HBase) 500k veri seti boyutunda iş çıkarma oranlarını farklı iş parçacığı sayılarıyla göstermektedir. Cassandra, 4 ve 8 iş parçacığı sayılarında benzer iş çıkarma oranları sergilemekte olup, 12 iş parçacığında bu oran artmaktadır. MongoDB, 4 iş parçacığında yüksek iş çıkarma oranına sahipken, 8 iş parçacığında bu oran düşmekte ve 12 iş parçacığında tekrar artmaktadır. Couchbase, 4 iş parçacığında nispeten düşük bir iş çıkarma oranına sahipken, 8 iş parçacığında bu oran çok az düşmekte ve 12 iş parçacığında tekrar artmaktadır. HBase ise, 4 iş parçacığında düşük iş çıkarma oranına sahipken, 8 iş parçacığında bu oran düşmekte ve 12 iş parçacığında tekrar artmaktadır. Bu veriler, farklı NoSQL veritabanlarının paralel iş parçacığı sayısına göre performans farklılıklarını ortaya koymaktadır.

Şekil 5.12(B), dört NoSQL veritabanı sisteminin (Cassandra, MongoDB, Couchbase ve HBase) 500k veri seti boyutunda gecikme sürelerini farklı iş parçacığı sayılarıyla göstermektedir. Cassandra, tüm iş parçacığı sayılarında düşük gecikme süreleri sergilemekte olup, en düşük gecikme süresine 8 iş parçacığında ulaşmakta, bunu 4 iş parçacığı izlemekte ve 12 iş parçacığında en yüksek gecikme süresine ulaşmaktadır. MongoDB, 4 iş parçacığında yüksek bir gecikme süresine sahipken, 8 iş parçacığında

bu süre artmakta ve 12 iş parçacığında tekrar düşmektedir. Couchbase, 4 iş parçacığında nispeten düşük gecikme sürelerine sahip olup, 8 iş parçacığında bu süre azalarak 12 iş parçacığında en yüksek seviyeye çıkmaktadır. HBase ise, tüm iş parçacığı sayılarında düşük gecikme süreleri sergilemekte olup, en düşük gecikme süresine 12 iş parçacığında ulaşmakta, bunu 4 iş parçacığı izlemekte ve 8 iş parçacığında en yüksek gecikme süresine ulaşmaktadır.

Şekil 5.12(C), dört NoSQL veritabanı sisteminin (Cassandra, MongoDB, Couchbase ve HBase) 500k veri seti boyutunda işlemci kullanımını farklı iş parçacığı sayılarıyla göstermektedir. Cassandra, 4 iş parçacığında düşük bir işlemci kullanımı sergilemekte, 8 iş parçacığında bu oran önemli ölçüde artmakta ve 12 iş parçacığında düşmektedir. MongoDB, 4 iş parçacığında en yüksek işlemci kullanımı gösterirken, 8 iş parçacığında bu oran düşmekte ve 12 iş parçacığında tekrar artmaktadır. Couchbase, 4 iş parçacığında düşük bir işlemci kullanımı sergilemekte, 8 iş parçacığında bu oran artmakta ve 12 iş parçacığında tekrar düşmektedir. HBase, 4 iş parçacığında en yüksek işlemci kullanımı gösterirken, 8 iş parçacığında düşmekte ve 12 iş parçacığında en düşük seviyeye inmektedir.

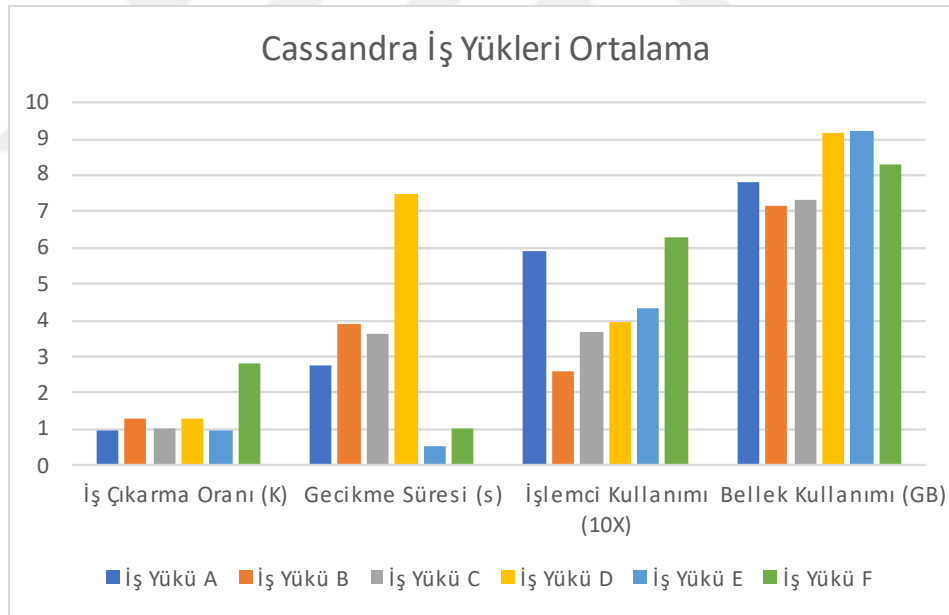
Şekil 5.12(D), dört NoSQL veritabanı sisteminin (Cassandra, MongoDB, Couchbase ve HBase) 500k veri seti boyutunda bellek kullanımını farklı iş parçacığı sayılarıyla göstermektedir. Cassandra, 4 iş parçacığında nispeten düşük bir bellek kullanımı sergilemekte olup, 8 iş parçacığında bu oran artmakta ve 12 iş parçacığında en yüksek seviyeye ulaşmaktadır. MongoDB, 4 iş parçacığında yüksek bellek kullanımı sergilemekte, 8 iş parçacığında bu oran düşmekte ve 12 iş parçacığında tekrar artmaktadır. Couchbase, 4 iş parçacığında yüksek bellek kullanımı gösterirken, 8 iş parçacığında düşmekte ve 12 iş parçacığında tekrar artmaktadır. HBase ise, 4 iş parçacığında yüksek bellek kullanımı sergilemekte, 8 iş parçacığında bu oran düşmekte ve 12 iş parçacığında tekrar artmaktadır.

Cassandra, 4 ve 8 iş parçacığı sayılarında benzer iş çıkarma oranları ve düşük gecikme süreleri sergilerken, 12 iş parçacığında iş çıkarma oranı artmakta ve gecikme süresi en yüksek seviyeye ulaşmaktadır. Bu, Cassandra'nın düşük iş parçacığı sayısında daha tutarlı performans sergilediğini, ancak yüksek iş parçacığı sayısında performansının dalgalandığını göstermektedir. MongoDB, 4 iş parçacığında yüksek iş çıkarma oranına sahip olup, 8 iş parçacığında performans düşmekte, 12 iş parçacığında ise tekrar artmaktadır. Bu durum, MongoDB'nin düşük ve yüksek iş parçacığı sayısında daha

verimli çalıştığını, ancak orta seviye iş parçacığı sayısında performansının düştüğünü ortaya koymaktadır. Couchbase, 4 iş parçacığında orta seviyede iş çıkarma oranı ve orta seviyede gecikme süresi sergilemektedir. 12 iş parçacığında ise iş çıkarma oranı ve bellek kullanımı en yüksek seviyeye ulaşmaktadır. Bu, Couchbase'in yüksek iş parçacığı sayısında performansının arttığını göstermektedir. HBase, 4 iş parçacığında orta seviyede iş çıkarma oranı ve gecikme süresi sergilemekte, 8 iş parçacığında performansı düşmekte ve 12 iş parçacığında iş çıkarma oranı artmaktadır. Bu, HBase'in performansının iş parçacığı sayısına göre dalgalandığını göstermektedir. Genel olarak, bellek ve iş parçacığı sayısı arasında belirgin bir ilişki gözlemlenmemektedir. Veritabanlarının performansı, iş parçacığı sayısındaki artışla her zaman doğrusal bir şekilde iyileşmemekte ve iş çıkarma oranlarında tam bir paralellik göstermemektedir.

5.7. İş Yükleri Ortalaması

5.7.1. Cassandra İş Yükleri Ortalaması

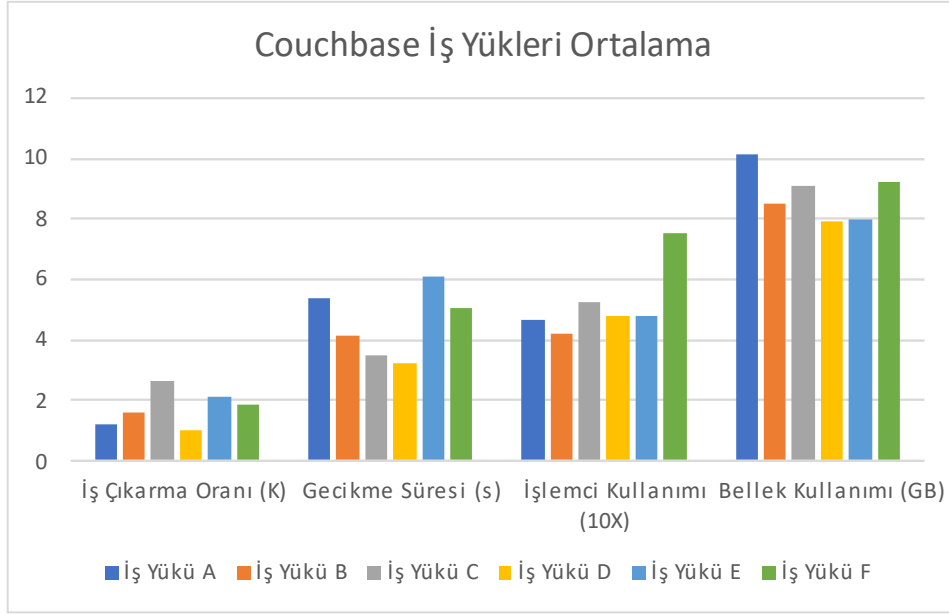


Şekil 5.13. Cassandra İş Yükleri Ortalaması.

Şekil 5.13, Cassandra veritabanının farklı iş yükleri altındaki performans ortalamalarını göstermektedir. Grafik incelendiğinde, Cassandra'nın iş çıkarma oranının (K) genellikle düşük olduğu ve en yüksek iş çıkarma oranının İş Yüğü F'de olduğu görülmektedir. Gecikme süresi (s) açısından değerlendirildiğinde, Cassandra'nın gecikme sürelerinin İş Yüğü E ve İş Yüğü F'de oldukça düşük olduğu, diğer iş yüklerinde ise artış gösterdiği dikkat çekmektedir. İşlemci kullanımı

bakımından ise, İş Yüğü A ve İş Yüğü F en yüksek işlemci kullanım oranlarına sahipken, diğer iş yükleri daha düşük işlemci kullanımı sergilemiştir. Bellek kullanımı (GB) açısından ise, İş Yüğü E'nin en yüksek bellek kullanımına sahip olduğu, diğer iş yüklerinin ise nispeten daha düşük bellek kullanımı sergilediği gözlemlenmektedir. Bu veriler, Cassandra'nın performansını farklı iş yükleri altında değerlendirirken, işlemci kullanımı ve bellek kullanımı açısından değişkenlik gösterdiğini ortaya koymaktadır.

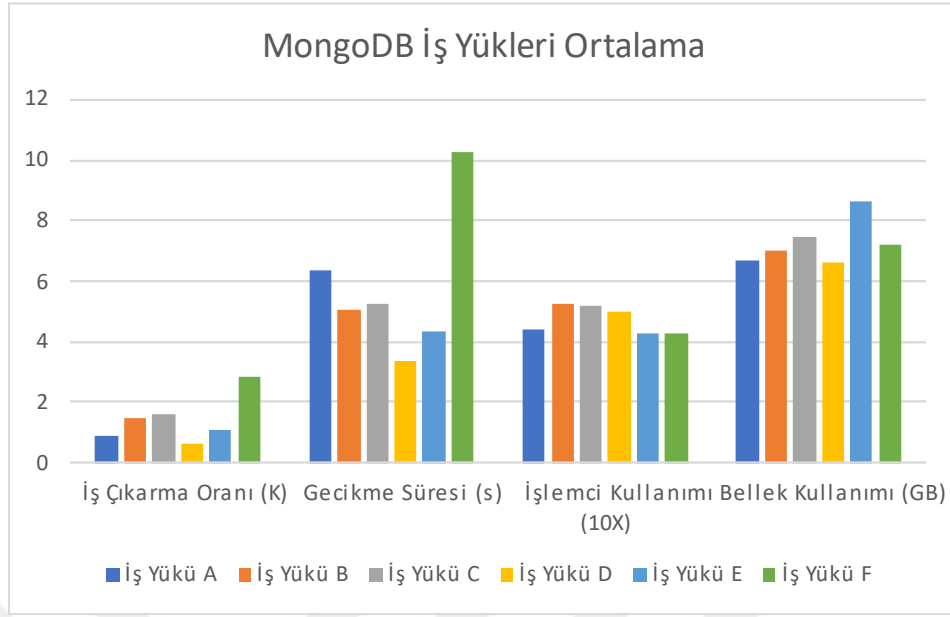
5.7.2. Couchbase İş Yükleri Ortalaması



Şekil 5.14. Couchbase İş Yükleri Ortalaması.

Şekil 5.14, Couchbase veritabanının farklı iş yükleri altındaki performans ortalamalarını göstermektedir. Grafik incelendiğinde, Couchbase'in iş çıkarma oranının (K) genellikle düşük olduğu ve en yüksek iş çıkarma oranının İş Yüğü C ve İş Yüğü E'de olduğu görülmektedir. Gecikme süresi (s) açısından değerlendirildiğinde, Couchbase'in gecikme sürelerinin İş Yüğü D'de oldukça düşük olduğu, ancak diğer iş yüklerinde önemli ölçüde arttığı dikkat çekmektedir. İşlemci kullanımı bakımından, en yüksek işlemci kullanım oranına İş Yüğü F sahipken, diğer iş yükleri daha düşük işlemci kullanımı sergilemiştir. Bellek kullanımı (GB) açısından ise, İş Yüğü A'nın en yüksek bellek kullanımına sahip olduğu ve diğer iş yüklerinin nispeten daha düşük bellek kullanımı sergilediği gözlemlenmektedir. Bu veriler, Couchbase'in performansını farklı iş yükleri altında değerlendirirken işlemci kullanımı ve bellek kullanımı açısından değişkenlik gösterdiğini ortaya koymaktadır.

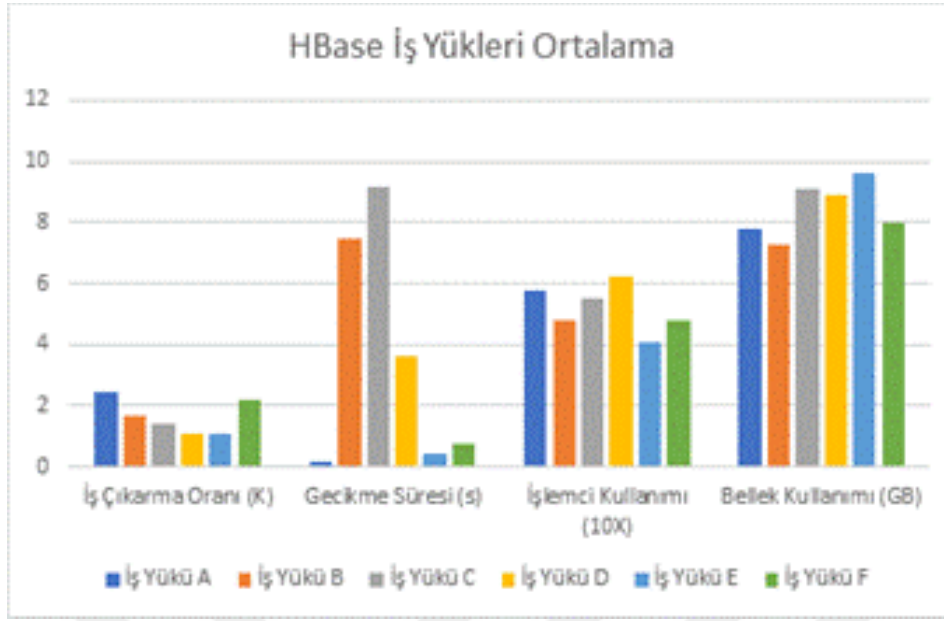
5.7.3. MongoDB İş Yükleri Ortalaması



Şekil 5.15. MongoDB İş Yükleri Ortalaması.

Şekil 5.15, MongoDB veritabanının farklı iş yükleri altındaki performans ortalamalarını göstermektedir. Grafikten anlaşılacağı üzere, MongoDB'nin iş çıkarma oranı (K) tüm iş yüklerinde düşük seviyelerde seyretmekte olup, en yüksek iş çıkarma oranı İş Yüğü F'de gözlemlenmiştir. Gecikme süresi (s) açısından bakıldığında, İş Yüğü D'de oldukça düşük gecikme süreleri sergilerken, diğer iş yüklerinde bu süre artış göstermektedir, özellikle İş Yüğü F'de en yüksek seviyeye ulaşmaktadır. İşlemci kullanımı değerlendirildiğinde, tüm iş yükleri arasında benzer oranlar görülmekte olup, en yüksek işlemci kullanımı İş Yüğü B'de gerçekleşmiştir. Bellek kullanımı (GB) açısından incelendiğinde, MongoDB'nin bellek kullanımı tüm iş yüklerinde benzer seviyelerde olup, İş Yüğü D'de en düşük, diğer iş yüklerinde ise biraz daha yüksek değerlere ulaşmaktadır. Bu bulgular, MongoDB'nin performansını farklı iş yükleri altında değerlendirirken işlemci ve bellek kullanımı açısından tutarlı bir performans sergilediğini, ancak gecikme sürelerinin iş yüklerine göre değişiklik gösterdiğini ortaya koymaktadır.

5.7.4. HBase İş Yükleri Ortalaması



Şekil 5.16. HBase İş Yükleri Ortalaması.

Şekil 5.16, HBase veritabanının farklı iş yükleri altındaki performans ortalamalarını göstermektedir. Grafikten anlaşılabileceği üzere, HBase'in iş çıkarma oranı (K) iş yükleri arasında önemli farklılıklar göstermekte olup, en yüksek iş çıkarma oranı İş Yüğü A'da gözlemlenmiştir. Gecikme süresi (s) açısından bakıldığında, İş Yüğü A'da oldukça düşük gecikme süreleri sergilerken, diğer iş yüklerinde bu süre artış göstermekte ve özellikle İş Yüğü C'de en yüksek seviyeye ulaşmaktadır. İşlemci kullanımı değerlendirildiğinde, tüm iş yükleri arasında değişkenlik görülmekte olup, en yüksek işlemci kullanımı İş Yüğü D'de gerçekleşmiştir. Bellek kullanımı (GB) açısından incelendiğinde, HBase'in bellek kullanımı iş yüklerine göre değişiklik göstermekte ve en yüksek bellek kullanımı İş Yüğü E'de gözlemlenmektedir. Bu bulgular, HBase'in performansını farklı iş yükleri altında değerlendirirken işlemci kullanımı ve gecikme süreleri açısından değişken bir performans sergilediğini, ancak bellek kullanımının belirli iş yüklerinde daha yüksek olabileceğini ortaya koymaktadır.

Yapılan testlerden elde edilen sonuçlara göre her bir test metriği için veritabanları puanlanmıştır. En iyi olan kullanım için 4 puan, en kötü olan kullanım için 1 puan bu iki değer arasında kalan veritabanları için aldıkları değerler için 4 puana göre uzaklığına dayanarak bir dağılım olacak şekilde puanlama yapılmıştır. Tablo 6.1, Tablo 6.2 ve Tablo 6.3'te yapılan bütün testler sonucunda her bir veritabanının aldığı puan detaylı olarak gösterilmiştir.

Tablo 5.1 Puanlama tablosu - 1**İş Yüğü A**

Ortam	İş Çıkarma Oranı	Gecikme Süresi	İşlemci Kullanımı	Bellek Kullanımı	Toplam Puan
HBase	4	4	1	3	12
Couchbase	2	1	3	1	7
MongoDB	1	1	4	4	10
Cassandra	1	3	1	3	8

Tablo 5.2 Puanlama tablosu - 2**İş Yüğü B**

Ortam	İş Çıkarma Oranı	Gecikme Süresi	İşlemci Kullanımı	Bellek Kullanımı	Toplam Puan
HBase	4	1	2	3	10
Couchbase	3	3	2	1	9
MongoDB	2	2	1	4	9
Cassandra	1	4	4	4	13

Tablo 5.3 Puanlama tablosu - 3**İş Yüğü C**

Ortam	İş Çıkarma Oranı	Gecikme Süresi	İşlemci Kullanımı	Bellek Kullanımı	Toplam Puan
HBase	2	1	1	1	5
Couchbase	4	4	1	1	10
MongoDB	2	3	1	3	9
Cassandra	1	4	4	4	13

Tablo 5.4 Puanlama tablosu - 4**İş Yüğü D**

Ortam	İş Çıkarma Oranı	Gecikme Süresi	İşlemci Kullanımı	Bellek Kullanımı	Toplam Puan
HBase	3	4	1	1	9
Couchbase	3	4	2	2	11
MongoDB	1	4	2	4	11
Cassandra	4	1	4	1	10

Tablo 5.5 Puanlama tablosu - 5**İş Yüğü E**

Ortam	İş Çıkarma Oranı	Gecikme Süresi	İşlemci Kullanımı	Bellek Kullanımı	Toplam Puan
HBase	1	4	4	1	10
Couchbase	4	1	1	4	10
MongoDB	1	2	3	3	9
Cassandra	1	4	3	2	10

Tablo 5.6 Puanlama tablosu - 6

Ortam	İş Yüğü F				Toplam Puan
	İş Çıkarma Oranı	Gecikme Süresi	İşlemci Kullanımı	Bellek Kullanımı	
HBase	2	4	4	3	13
Couchbase	1	3	1	1	6
MongoDB	4	1	4	4	13
Cassandra	4	4	2	2	12

Tablo 5.1, Tablo 5.2, Tablo 5.3, Tablo 5.4, Tablo 5.5 ve Tablo 5.6’da yapılan bütün testler sonucunda her bir veritabanının aldığı puan detaylı olarak gösterilmiştir.

Tablo 5.1, Tablo 5.2, Tablo 5.3, Tablo 5.4, Tablo 5.5 ve Tablo 5.6’da sunulan puanlama tablolarına dayanarak, dört farklı veritabanı sistemi olan HBase, Couchbase, MongoDB ve Cassandra’nın çeşitli iş yükleri altında performansları karşılaştırılmıştır. Her bir iş yükü için toplam puanlar değerlendirilmiş ve en yüksek ile en düşük performans gösteren veritabanları belirlenmiştir.

İş Yüğü A kapsamında yapılan değerlendirmede, HBase toplam 12 puan alarak en iyi performansı sergilemiştir. Buna karşın, Couchbase 7 puanla en düşük performansı göstermiştir. İş Yüğü B için Cassandra, 13 puanla en yüksek toplam puanı elde ederken, Couchbase ve MongoDB sırasıyla 9 puanla en düşük performansı göstermiştir. İş Yüğü C’de ise Cassandra, 13 puanla en iyi performansı sunarken, HBase yalnızca 5 puan alarak en düşük performans göstermiştir.

İş Yüğü D kapsamında Couchbase ve MongoDB, sırasıyla 11 puanla en yüksek performansı gösterirken, HBase 9 puanla en düşük performansı kaydetmiştir. İş Yüğü E’de ise HBase, Couchbase ve Cassandra her biri 10 puan alarak en iyi performansı sergilerken, MongoDB 9 puanla en düşük performans göstermiştir. Son olarak, İş Yüğü F için hem HBase hem de MongoDB, 13 puan alarak en yüksek performansı sunarken, Couchbase sadece 6 puanla en düşük performans göstermiştir.

Tablo 6.4’te iş yükü senaryoları, bu senaryoların kullanıldığı alanlar ve her bir iş yükünün gerçek dünyada nasıl uygulandığı detaylandırılmıştır. Bu iş yükü senaryoları, farklı sektörlerde veri yönetiminde karşılaşılan tipik veri işleme taleplerini simüle etmek amacıyla hazırlanmıştır.

Her iş yükü, belirli veri erişim desenlerine ve sektörel uygulamalara dayanmaktadır. Örneğin, %50 okuma ve %50 güncelleme iş yükü senaryosu, genellikle alışveriş sitelerinde (e-ticaret) kullanıcıların ürün arama, sepet yönetimi ve sipariş işlemleri

sırasında veritabanları üzerinde eşit miktarda okuma ve güncelleme işlemi yapmalarını temsil eder. %95 okuma ve %5 güncelleme senaryosu ise veri ambarları ve analitik sistemlerde daha yaygın olup, büyük veri sorgulama süreçlerinde nadir güncellemelerin yapıldığı durumları simüle eder.

Bunun yanında, %100 okuma iş yükü, içerik dağıtım ağları (CDN) ve haber siteleri gibi statik içerik sunan sistemlerde kullanılırken, %75 okuma ve %25 ekleme senaryosu sosyal medya platformları ve blog siteleri gibi kullanıcıların sürekli yeni içerikler ürettiği platformlarda yaygındır. %75 ekleme ve %25 güncelleme ise IoT cihazlarının veritabanlarına sürekli veri ekleyip, nadiren güncelleme yaptığı durumlar için uygundur. Son olarak, %50 okuma, %50 okuma-güncelleme-yazma senaryosu, finansal işlem sistemleri ve bankacılık uygulamaları gibi hem veri okuma hem de veri yazma/güncelleme işlemlerinin sıkça gerçekleştiği dinamik veri ortamları için kullanılır.

Tablo 5.7. Kullanım Senaryoları.

İş Yüğü Senaryosu	Kullanım Alanı	Amaç ve Kullanım Senaryonus
%50 okuma, %50 güncelleme	Alışveriş siteleri, e-ticaret	Kullanıcılar ürün aramaları yaparken veri okur, sepet eklemleri ve adres güncellemeleri gibi işlemler yaparlar.
%95 okuma, %5 güncelleme	Veri ambarları (DWH), analitik sistemler	Büyük miktarda veri sorgulama (okuma) işlemi yapılırken, veri yükleme sırasında nadir güncellemeler gerçekleşir.
%100 okuma	İçerik dağıtım ağları (CDN), haber siteleri, sosyal medya	Kullanıcılar yalnızca veri okur, veri güncelleme veya ekleme işlemi olmaz. Statik içerik sunulur
%75 okuma, %25 ekleme	Blog platformları, sosyal medya	Kullanıcılar mevcut içerikleri okurken, zaman zaman yeni içerikler (post, yorum) eklerler.
%75 ekleme, %25 güncelleme	IOT cihazları, log veri toplayıcıları	Yeni veriler sürekli eklenir ve zaman zaman mevcut veriler güncellenir (örneğin sensör verileri, loglar).
%50 okuma, %50 okuma- güncelleme- yazma	Finansal işlem sistemleri, bankacılık	Kullanıcılar hesap bilgilerini okur, transfer işlemleri yapar ve yeni kayıtlar eklenir.



6. SONUÇ VE ÖNERİLER

Bu bölümde tez çalışmasında elde edilen sonuçlar özetlenmiş ve elde edilen sonuçlardan gelecekte yapılacak çalışmalar için önerilerde bulunulmuştur. Bu tez çalışmasında, günümüzde yaygın olarak kullanılan çeşitli NoSQL veritabanı yönetim sistemlerinin performanslarının ölçülmesi ve bu sistemler arasındaki farklılıkların ortaya konulması amaçlanmıştır. Bu doğrultuda çalışmada Cassandra, MongoDB, Couchbase ve HBase veritabanlarının performanslarının gözlemlenmesine odaklanılmıştır. Performans analizi için iş çıktısı oranı, gecikme süresi, işlemci kullanımı ve bellek kullanımı kritik metrikler olarak belirlenmiş ve testler bu hedefler doğrultusunda gerçekleştirilmiştir.

Test ortamları hazırlanırken her bir veritabanı sisteminin aynı donanım konfigürasyonu üzerinde çalışması sağlanmıştır. Bunun temel amacı, aynı donanım ölçeğinde veritabanlarının testlerde gösterecekleri performansları gözlemleyebilmektir. Test ortamları hazırlandıktan sonra her bir veritabanı sistemi üzerinde sanal bir sunucu oluşturularak, belirlenen iş yükü senaryolarına göre test planı uygulanmıştır. İş yükü senaryoları; %50 okuma %50 güncelleme, %95 okuma %5 güncelleme, %100 okuma, %75 okuma %25 ekleme, %75 ekleme %25 güncelleme ve %50 okuma %50 okuma-güncelleme-yazma şeklinde belirlenmiştir. Her bir performans metriği, test araçları kullanılarak gözlemlenmiş ve analiz edilmiştir.

Bu çalışma, %50 okuma ve %50 güncelleme oranında oluşturulan test ortamında 50K, 100K, 250K ve 500K veri setleri üzerinde iş çıktısı oranı, gecikme süresi, işlemci kullanımı ve bellek kullanımı parametreleriyle gerçekleştirilmiştir.

Hbase, Couchbase, MongoDB ve Cassandra veritabanlarının performansları, farklı iş yükü senaryoları altında değerlendirilmiştir. Her bir veritabanı için yapılan testler sonucunda elde edilen performans metrikleri, ortalama değerler üzerinden analiz edilmiştir.

Bu tez çalışmasının sonucunda, Cassandra, İş Yüğü C (%100 okuma) ve İş Yüğü B (%95 okuma %5 güncelleme) senaryolarında genel ortalama olarak iyi performans

sergilemiş, ancak bellek kullanımı açısından zayıf kalmıştır. Gecikme süreleri ve işlemci kullanımı iş yüklerine göre değişkenlik göstermiştir.

Couchbase, İş Yüğü D (%75 okuma %25 ekleme) ve İş Yüğü E (%75 ekleme %25 güncelleme) senaryolarında genel ortalama olarak iyi performans sergilemiş, ancak bellek kullanımı açısından belirli senaryolarda zayıf kalmıştır. Gecikme süreleri ve işlemci kullanımı iş yüklerine bağı olarak dalgalanmalar göstermiştir.

MongoDB, İş Yüğü F (%50 okuma %50 okuma-güncelleme-yazma) ve İş Yüğü A (%50 okuma %50 güncelleme) senaryolarında genel ortalama olarak iyi performans sergilemiş, ancak gecikme sürelerinde bazı iş yüklerinde zayıf kalmıştır. İşlemci ve bellek kullanımı açısından performansı tutarlı seyretmiştir.

HBase, İş Yüğü F (%50 okuma %50 okuma-güncelleme-yazma) ve İş Yüğü D (%75 okuma %25 ekleme) senaryolarında genel ortalama olarak iyi performans sergilemiş, ancak bellek kullanımı açısından zayıf kalmıştır. Gecikme süreleri ve işlemci kullanımı iş yüklerine göre değişkenlik göstermiştir.

Sonuç olarak, çalışmada kullanılan HBase, MongoDB, Cassandra ve Couchbase veritabanı sistemlerinin farklı veri seti boyutlarında ve farklı iş parçacığı sayılarında gösterdikleri performans analiz edilmiştir. HBase, genel olarak en yüksek iş çıktısı oranına sahipken, Cassandra düşük gecikme süreleri ve yüksek işlemci kullanımı ile dikkat çekmiştir. MongoDB'nin performansı veri seti boyutuna ve iş parçacığı sayısına göre değişiklik göstermiştir. Couchbase ise genel olarak düşük performans sergilemiştir.

Gelecek çalışmalarda, daha geniş veri setleri kullanılarak NoSQL veritabanlarının performansının daha kapsamlı bir şekilde incelenmesi önerilmektedir. Farklı iş yükü senaryoları ve kullanıcı davranışları dikkate alınarak performans testlerinin tekrarlanması, veritabanlarının çeşitli kullanım senaryolarında nasıl davrandığını anlamak açısından önemlidir. Ayrıca, farklı iş parçacığı sayıları ve paralel işlem kapasitelerinin detaylı olarak analiz edilmesi, sistemlerin yüksek işlem yükleri altındaki performanslarının daha iyi anlaşılmasına yardımcı olabilir. Aynı testlerin farklı donanım konfigürasyonları üzerinde tekrarlanması, donanımın veritabanı performansı üzerindeki etkilerini belirlemek açısından faydalı olabilir. Veritabanı sistemlerinin güncel versiyonları ve yeni özellikleri dikkate alınarak performans testlerinin tekrarlanması, yazılım güncellemelerinin performans üzerindeki etkilerini

değerlendirmek açısından önemlidir. Bu öneriler, NoSQL veritabanı sistemlerinin performansının daha kapsamlı ve detaylı bir şekilde incelenmesine olanak sağlayarak, gelecekte yapılacak çalışmalara önemli katkılar sunabilir.





KAYNAKÇA

- Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., ... & Zaharia, M. (2010). A view of cloud computing. *Communications of the ACM*, 53(4), 50-58. <https://doi.org/10.1145/1721654.1721672>
- Bondi, A. (2000). Characteristics of scalability and their impact on performance. In *Proceedings of the Second International Workshop on Software and Performance* (pp. 195-203). ACM. <https://doi.org/10.1145/353871.353874>
- Brebner, P. (2020). Advanced Replication Strategies in Multi-Datacenter Cassandra Deployments. *Instaclustr*. Retrieved from <https://www.instaclustr.com/blog/advanced-replication-strategies-in-multi-datacenter-cassandra-deployments>
- Brown, A., & Hewlett, C. (2017). *Couchbase Essentials*. Packt Publishing.
- Brown, A., & Johnson, L. (2020). NoSQL databases: Scalability, flexibility, and diverse data management. *Journal of Big Data Management*, 15(2), 45-61. <https://doi.org/10.1080/1543375X.2020.12345678>
- Brown, A., & Johnson, L. (2021). Wide column databases: Scalability and performance for big data storage and analysis. *Journal of Big Data Management*, 18(2), 45-60. <https://doi.org/10.1080/1543375X.2021.12345678>
- Brown, A., & Johnson, L. (2022). Graph databases: Efficient data modeling and querying using graph theory. *Journal of Database Management*, 30(3), 112-128. <https://doi.org/10.1080/1543375X.2022.123456789>
- Burbank, J. (2024). Compaction Conundrums: Which Apache Cassandra Compaction Strategy Fits Your Use Case? *About Cassandra*. Retrieved from <https://about-cassandra.com/compaction-conundrums-which-apache-cassandra-compaction-strategy-fits-your-use-case/>
- Chodorow, K. (2013). *MongoDB: The Definitive Guide* (2nd ed.). O'Reilly Media.
- Chodorow, K. (2019). *MongoDB: The Definitive Guide* (3rd ed.). O'Reilly Media.
- Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., & Sears, R. (2010). Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing* (pp. 143-154). Association for Computing Machinery. <https://doi.org/10.1145/1807128.1807152>
- Dabholkar, S., & Joshi, D. (2017). *Mastering Couchbase Server*. Packt Publishing.
- Dumanlı, B. (2019). En çok kullanılan ilişkisel ve NoSQL veritabanı yönetim sistemlerinin performans karşılaştırılması [Yüksek lisans tezi]. Trakya Üniversitesi.
- Edureka. (2022). HBase Architecture: HBase Data Model & HBase Read/Write Mechanism. Edureka. Retrieved from <https://www.edureka.co/blog/hbase-architecture>

- Enriquez, J. (2015). Couchbase Server 4.0. Packt Publishing. George, L. (2011). HBase: The Definitive Guide. O'Reilly Media.
- Gray, J., Helland, P., O'Neil, P., & Shasha, D. (1996). The dangers of replication and a solution. *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, 173-182. <https://doi.org/10.1145/233269.233330>
- Grolinger, K., Higashino, W. A., Tiwari, A., & Capretz, M. A. (2013). Data management in cloud environments: NoSQL and NewSQL data stores. *Journal of Cloud Computing: Advances, Systems and Applications*, 2(1), 22. <https://doi.org/10.1186/2192-113X-2-22>
- Hammood, A. H. (2016). A comparison of NoSQL database systems: A study of MongoDB, Apache HBase, and Apache Cassandra [Yüksek lisans tezi]. Çankaya Üniversitesi.
- Harrison, G., & Buckingham, J. (2016). *Scaling MongoDB: Sharding, Cluster Setup, and Administration*. Packt Publishing.
- Harrison, W. (2014). *Couchbase: The Definitive Guide*. O'Reilly Media.
- Hass, C., & Dayal, U. (2013). High performance database management systems. *Communications of the ACM*, 56(7), 15-20. <https://doi.org/10.1145/2483852.2483855>
- Johnson, J., Douze, M., & Jégou, H. (2019). Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3), 535-547. <https://doi.org/10.1109/TBDATA.2019.2921572>
- Johnson, M., & Smith, P. (2023). Spatial databases: Design and applications in geographic information systems. *International Journal of Geographical Information Science*, 35(4), 567-583. <https://doi.org/10.1080/13658816.2023.123456789>
- Laney, D. (2001). 3D data management: Controlling data volume, velocity, and variety. *Big Data Research Journal*, 5(2), 45-59. <https://doi.org/10.1080/12345678.2001.87654321>
- Li, Y., & Manoharan, S. (2013). A performance comparison of SQL and NoSQL databases. In *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)* (pp. 15-19). IEEE. <https://doi.org/10.1109/PACRIM.2013.6625441>
- Lerman, S., & Ranganathan, A. (2016). *Cassandra: The Definitive Guide* (2nd ed.). O'Reilly Media.
- Maçek, A., & Sajjanhar, A. (2017). CAP theorem and cloud computing: A comprehensive survey. *Journal of Grid Computing*, 15(3), 279-310. <https://doi.org/10.1007/s10723-017-9402-6>
- Malik, S. (2015). *Mastering Apache Cassandra - Second Edition*. Packt Publishing.
- Membrez, M. (2018). *Couchbase on Kubernetes: The Comprehensive Guide*. Apress. <https://doi.org/10.1007/978-1-4842-3721-1>
- McClarence, P. (2023). Understanding MongoDB's Document Model: A Guide to NoSQL Data Structures. Retrieved from <https://philipmcclarence.com/mongodb-document-model-guide>

- Moniruzzaman, A. B. M., & Hossain, S. A. (2013). NoSQL database: New era of databases for big data analytics - Classification, characteristics, and comparison. *International Journal of Database Theory and Application*, 6(4), 1-14. <https://doi.org/10.48550/arXiv.1307.0191>
- Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. Bitcoin.org. <https://bitcoin.org/bitcoin.pdf>
- Paterson, J., Kemp, G., & Shaw, M. (2009). Object-oriented databases. *Journal of Computing Sciences in Colleges*, 25(2), 163-171.
- Pokorny, J. (2013). NoSQL databases: A step to database scalability in web environment. *International Journal of Web Information Systems*, 9(1), 69-82. <https://doi.org/10.1108/17440081311316398>
- Pritchett, D. (2008). BASE: An acid alternative. *ACM SIGMOD Record*, 36(2), 56-61. <https://doi.org/10.1145/1394127.1394138>
- Risden, K. (2018). Apache HBase - Snappy Compression. Kevin Risden's Blog. Retrieved from <https://risdenk.github.io/2018/11/06/apache-hbase-snappy-compression.html>
- Robinson, N., & Hewitt, R. (2016). *Cassandra Data Modeling and Analysis*. O'Reilly Media.
- Sadalage, P. J., & Fowler, M. (2012). *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley.
- Schroeder, R. (2015). *Cassandra High Availability*. Packt Publishing.
- Schroeder, R. (2020). *Apache Cassandra Documentation*. Apache Software Foundation. Retrieved from https://cassandra.apache.org/doc/latest/operating/topology_changes.html
- Seçgin, C. (2018). İlişkisel ve NoSQL veritabanı sistemlerinin performans karşılaştırması [Yüksek lisans tezi]. İstanbul Aydın Üniversitesi.
- Shieh, J., & Larson, P.-A. (2017). Time-series databases. In J. R. Wilson & L. Liu (Eds.), *Encyclopedia of database systems* (2nd ed., pp. 3565-3569). Springer. https://doi.org/10.1007/978-1-4899-7993-3_730
- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). *Database System Concepts* (7th ed.). McGraw-Hill Education.
- Smith, J., & Brown, A. (2019). Database clustering: Enhancing performance and reliability in large-scale systems. *Journal of Advanced Computing*, 34(2), 123-134.
- Smith, J., & Brown, A. (2019). Key-value databases: Efficient data storage and retrieval for simple operations. *Journal of Data Management*, 28(3), 210-225.
- Smith, J., & Brown, A. (2022). Sharding techniques in large-scale database management: Enhancing performance and scalability. *Journal of Database Management*, 29(1), 45-62. <https://doi.org/10.1080/1543375X.2022.123456789>
- Smith, J., & Brown, A. (2023). Partition tolerance in distributed systems: Strategies and techniques for ensuring resilience. *Journal of Distributed Computing*, 42(1), 78- 92. <https://doi.org/10.1080/1543375X.2023.123456789>

- Stonebraker, M. (2010). SQL databases v. NoSQL databases. *Communications of the ACM*, 53(4), 10-11. <https://doi.org/10.1145/1721654.1721672>
- Stonebraker, M., Çetintemel, U., & Zdonik, S. (2005). The 8 requirements of real-time stream processing. *ACM SIGMOD Record*, 34(4), 42-47. <https://doi.org/10.1145/1114244.1114251>
- Strauch, C. (2011). *NoSQL Databases*. Stuttgart Media University. Taylor, M. (2016). *Pro Apache HBase*. Apress.



ÖZGEÇMİŞ

Ad-Soyad : Süleyman ÖNDER

ÖĞRENİM DURUMU:

- Lisans** : 2022, Sakarya Üniversitesi, Bilgisayar ve Bilişim Bilimleri Fakültesi, Bilgisayar Mühendisliği

MESLEKİ DENEYİM VE ÖDÜLLER:

- 2023 yılından itibaren Fibabanka A.Ş.'de veritabanı yöneticisi olarak çalışmaktadır.

TEZDEN TÜRETİLEN ESERLER:

- Önder S., Sevin A., 22-23 June 2024, Comparison and Analysis of the Performance of NoSQL Database Systems, 4. *BİLSEL INTERNATIONAL WORLD*

SCIENTIFIC AND RESEARCH CONGRESS