T.C. İSTANBUL KÜLTÜR ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

MİKROSERVİS EKOSİSTEMİNDE SERVİS DURUM YÖNETİMİ

YÜKSEK LİSANS TEZİ
FURKAN KARATAŞ
1800000473

Anabilim Dalı: Bilgisayar Mühendisliği

Programı: Bilgisayar Mühendisliği

Tez Danışmanı: Doç. Dr. Akhan Akbulut

MAYIS 2021

T.C. İSTANBUL KÜLTÜR ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

MİKROSERVİS EKOSİSTEMİNDE SERVİS DURUM YÖNETİMİ

YÜKSEK LİSANS TEZİ
FURKAN KARATAŞ
1800000473

Anabilim Dalı: Bilgisayar Mühendisliği
Programı: Bilgisayar Mühendisliği

Tez Danışmanı: Doç. Dr. Akhan Akbulut
Jüri Üyesi: Prof. Dr. Banu Diri
Jüri Üyesi: Öğr. Üyesi Fatma Patlar Akbulut

MAYIS 2021

ÖNSÖZ

"Mikroservis Ekosisteminde Servis Durum Yönetimi" adlı tez çalışmamda bilgisiyle, deneyimiyle, araştırmalarıma olan katkısıyla ve çalışmalarımı yönlendirmesiyle desteğini sağlayan değerli tez danışmanım Doç. Dr. Akhan AKBULUT'a çok teşekkür ederim. Her zaman, her durumda koşulsuz sevgilerini ve desteklerini esirgemeyen anneme, babama, kardeşlerime ve kız arkadaşıma teşekkür ederim.

Mayıs 2021

Furkan KARATAŞ

İÇİNDEKİLER

ÖNSÖZ ..... **ii**

İÇİNDEKİLER ..... **iii**

ŞEKİL LİSTESİ ..... **vi**

TABLO LİSTESİ ..... **viii**

KISALTMALAR VE TERİMLER ..... **ix**

ÖZET ..... **x**

ABSTRACT ..... **xii**

1. GİRİŞ **1**

1.1. Problem Tanımı 2

1.2. Literatüre Katkıları 2

1.3. Tezin Organizasyonu 3

2. ÖN BİLGİLER VE LİTERATÜR ARAŞTIRMASI **4**

2.1. Ön Bilgiler 4

2.1.1. Mikroservis Nedir? 4

2.1.2. Mikroservis Ekosistemi 4

2.1.3. Mikroservis Mimarisinde Ön Bellek Yapıları 5

2.1.4. En Son Kullanılan Ön Bellek Yöntemi (LRU) 6

2.1.5. Dağıtılmış Ön Bellek 8

2.2. Literatür Araştırması 10

3. YÖNTEM **12**

3.1. Gereksinim Analizi 12

3.1.1. Sistem Gereksinimleri 12

3.1.1.1. Çalışma Ortamı Gereksinimleri 12

3.1.1.2. Yazılım Gereksinimleri 13

3.2. Kullanılan Teknolojiler 13

3.2.1. Geliştirme Platformları 13

4. ÖNERİLEN YAKLAŞIM: MİKROSERVİS YAPILARINDA ÖN BELLEK YÖNTEMİ 15

4.1. Modeller 15

4.1.1. Gömülü Tasarım Kalıbı 16

4.1.2. Gömülü Dağıtılmış Tasarım Kalıbı 17

4.1.3. İstemci-Sunucu Tasarım Kalıbı 18

4.1.4. Bulut Tasarım Kalıbı 19

4.1.5. Sepet Tasarım Kalıbı 21

4.1.6. Ters Vekil Sepet Tasarım Kalıbı 23

5. BULGULAR 24

5.1. Gömülü Tasarım Kalıbı ve Gömülü Dağıtılmış Tasarım Kalıbının Kıyaslanması 24

Test 1 25

Test 2 27

Test 3 29

Test 4 31

Test 5 33

Test 6 35

5.2. İstemci-Sunucu Tasarım Kalıbı ile Bulut (Cloud) Tasarım Kalıbının Kıyaslanması 37

Test 7 38

Test 8 41

Test 9 43

Test 10 45

Test 11 47

Test 12 49

5.3. Sepet Tasarım Kalıbının Gömülü Dağıtılmış ve İstemci-Sunucu Tasarım Kalıplarıyla Kıyaslanması 52

5.3.1. Tüm Tabloyu Sorgulayan Test Sonuçlarının Karşılaştırılması 53

5.3.2. Tek Bir Demedi (Tuple) Sorgulayan Test Sonuçlarının Karşılaştırılması… 54

6. TARTIŞMA 55

6.1. Geçerliliğe Yönelik Şartlar ve Tehditler 56

7. SONUÇLAR 57

KAYNAKÇA 59

ŞEKİL LİSTESİ

Şekil 2.1 - Ön belleğe obje ekleme işlemi 6
Şekil 2.2 - Ön Bellekten obje çağırma işlemi 7
Şekil 2.3 - İlişkisel Veri Tabanı Modeli Sistemi – RDBMS (a) ve Hafıza Veri Şebekesi Sistemi – IMDG (b) 9
Şekil 4.1 – Gömülü Tasarım Kalıbı 16
Şekil 4.2 – Servis Yapısının Oluşturulması 16
Şekil 4.3 – Gömülü Dağıtılmış Tasarım Kalıbı 17
Şekil 4.4 – Dağıtılmış Servis Yapısının Oluşturulması 17
Şekil 4.5 – İstemci-Sunucu Tasarım Kalıbı 18
Şekil 4.6 – Ön Bellek Yapılandırma Sınıfı 18
Şekil 4.7 – Bulut Tasarım Kalıbı 19
Şekil 4.9 – Bulut Konfigürasyon Sınıfı 20
Şekil 4.10 – Sepet Tasarım Kalıbı 21
Şekil 4.11 – Sepet Tasarım Kalıbı Hazelcast Konfigürasyon Yapısı 21
Şekil 4.12 – Hazelcast Dağıtım Sınıfı 22
Şekil 4.13 – Hazelcast Pod Durum Takibi 23
Şekil 4.14 – Ters Vekil Sepet Tasarım Kalıbı 23
Şekil 5.1 – Gömülü Tasarım Kalıbı ve Gömülü Dağıtılmış Tasarım Kalıbı için 10 Kullanıcıyla Yapılan Tüm Tabloyu Sorgulama Sonuçları 25
Şekil 5.2 – Gömülü Tasarım Kalıbı ve Gömülü Dağıtılmış Tasarım Kalıbı için 100 Kullanıcıyla Yapılan Tüm Tabloyu Sorgulama Sonuçları 27
Şekil 5.3 – Gömülü Tasarım Kalıbı ve Gömülü Dağıtılmış Tasarım Kalıbı için 1000 Kullanıcıyla Yapılan Tüm Tabloyu Sorgulama Sonuçları 29
Şekil 5.4 – Gömülü Tasarım Kalıbı ve Gömülü Dağıtılmış Tasarım Kalıbı için 10 Kullanıcıyla Yapılan Tek Bir Demedi Sorgulayan Sonuçlar 31
Şekil 5.5 – Gömülü Tasarım Kalıbı ve Gömülü Dağıtılmış Tasarım Kalıbı için 100 Kullanıcıyla Yapılan Tek Bir Demedi Sorgulayan Sonuçlar 33
Şekil 5.6 – Gömülü Tasarım Kalıbı ve Gömülü Dağıtılmış Tasarım Kalıbı için 1000 Kullanıcıyla Yapılan Tek Bir Demedi Sorgulayan Sonuçlar 35
Şekil 5.7 – İstemci-Sunucu Tasarım Kalıbı ve Bulut Tasarım Kalıbı için 10 Kullanıcıyla Yapılan Tüm Tabloyu Sorgulayan Sonuçlar 38

Şekil 5.8 – Bulut Tasarım Kalıbındaki Küme Yapısının Ram Bilgileri 40
Şekil 5.9 – İstemci-Sunucu Tasarım Kalıbı ve Bulut Tasarım Kalıbı için 100 Kullanıcıyla Yapılan Tüm Tabloyu Sorgulayan Sonuçlar 41
Şekil 5.10 – Bulut Tasarım Kalıbının Küme Yapısının Ram Bilgileri 42
Şekil 5.11 – İstemci-Sunucu Tasarım Kalıbı ve Bulut Tasarım Kalıbı için 1000 Kullanıcıyla Yapılan Tüm Tabloyu Sorgulayan Sonuçlar 43
Şekil 5.12 – İstemci-Sunucu Tasarım Kalıbı ve Bulut Tasarım Kalıbı için 10 Kullanıcıyla Yapılan Tek Bir Demedi Sorgulayan Sonuçlar 45
Şekil 5.13 – İstemci-Sunucu Tasarım Kalıbı ve Bulut Tasarım Kalıbı için 100 Kullanıcıyla Yapılan Tek Bir Demedi Sorgulayan Sonuçlar 47
Şekil 5.14 – İstemci-Sunucu Tasarım Kalıbı ve Bulut Tasarım Kalıbı için 1000 Kullanıcıyla Yapılan Tek Bir Demedi Sorgulayan Sonuçları 49
Şekil 5.14 – Bulut Tasarım Kalıbı Küme Yapısının Ram Bilgileri 50

TABLO LİSTESİ

Tablo 5.1 – Gömülü Tasarım Kalıbı ve Gömülü Dağıtılmış Tasarım Kalıbı için Tüm Tabloyu Sorgulayan Karşılaştırma 30
Tablo 5.2 – Gömülü Tasarım Kalıbı ve Gömülü Dağıtılmış Tasarım Kalıbı için Tek Bir Demedi Sorgulayan Karşılaştırma 36
Tablo 5.3 – İstemci-Sunucu Tasarım Kalıbı ve Bulut Tasarım Kalıbı için Tüm Tabloyu Sorgulayan Karşılaştırma 44
Tablo 5.4 – İstemci-Sunucu Tasarım Kalıbı ve Bulut Tasarım Kalıbı için Tek Bir Demedi Sorgulayan Karşılaştırma 51
Tablo 5.5 – Sepet Tasarım Kalıbı ve Diğer Tasarım Kalıpları için Tüm Tabloyu Sorgulayan Karşılaştırma 53
Tablo 5.6 – Sepet Tasarım Kalıbı ve Diğer Tasarım Kalıpları için Tek Bir Demedi Sorgulayan Karşılaştırma 54

KISALTMALAR VE TERİMLER

DRDS : Dağıtılmış İlişkisel Veritabanı Sistemi
IMDG : In-Memory Data Grid – Hafıza Veri Şebekesi
CRM : Customer Relationship Management – Müşteri İlişkileri Yönetimi
LRU : Least Recently Used – En Son Kullanılan Ön Bellek Yöntemi
JRE : Java Runtime Environment – Java Çalıştırma Ortamı
JDK : Java Development Kit – Java Geliştirme Kiti
TTL : Time to Live – Ön Bellek Yaşam Süresi
FIFO : First-In-First-Out – İlk Giren İlk Çıkar Ön Bellek Yöntemi
LIFO : Last-In-First-Out – Son Giren İlk Çıkar Ön Bellek Yöntemi
MRU : Most-Recently-Used – En Çok Kullanılan Ön Bellek Yöntemi
RDBMS : İlişkisel Veri Tabanı Model Sistemi
API : Application Programming Interface-Uygulama Programlama Arayüzü
RAM : Random Access Memory – Rastgele Erişimli Bellek
JVM : Java Virtual Machine – Java Sanal Makinesi
HttpRequest : Http İsteği
Load Balancer : Yük Dengeleyici
Embedded Pattern : Gömülü Tasarım Kalıbı
Embedded Distributed Pattern : Gömülü Dağıtılmış Tasarım Kalıbı
Client-Server Pattern : İstemci-Sunucu Tasarım Kalıbı
Cloud Pattern : Bulut Tasarım Kalıbı
Sidecar Pattern : Sepet Tasarım Kalıbı
Reverse Proxy Sidecar Pattern : Ters Vekil Sepet Tasarım Kalıbı

Üniversite	**:**	**T.C. İstanbul Kültür Üniversitesi**
Enstitü	**:**	**Lisansüstü Eğitim Enstitüsü**
Anabilim Dalı	**:**	**Bilgisayar Mühendisliği**
Program	**:**	**Bilgisayar Mühendisliği**
Tez Danışmanı	**:**	**Doç. Dr. Akhan AKBULUT**
Tez Türü ve Tarihi	**:**	**Yüksek Lisans – Mayıs 2021**

ÖZET

MİKROSERVİS EKOSİSTEMİNDE SERVİS DURUM YÖNETİMİ

Günümüzde mikroservis mimarisinin popülaritesi artmakta ve işletmeler tarafından tercih edilmektedir. Ağırlıklı olarak yazılım, finans, bankacılık, danışmanlık, e-ticaret ve telekomünikasyon sektörlerinde kullanılmaktadır. Mikroservis yapısının avantajı farklı teknolojileri bir arada kullanarak yapının esnek ve ölçeklendirilebilir hale getirilmesidir. Önerdiğimiz proje kapsamında, mikroservis mimarilerinin farklı tasarım kalıplarındaki ön bellek yapılarının kurgulanması, yerleştirilmesi ve mikroservislerin durum yönetimiyle ilgili süreçlerinde yardımcı olacak bir sistem geliştirilmiştir.

Geliştirilen sistem olan Sepet Tasarım Kalıbı (Sidecar Pattern) geçmişten günümüze kullanılan Gömülü, Gömülü Dağıtılmış, İstemci-Sunucu, Bulut tasarım kalıplarına alternatif bir yöntem olarak kullanılması amaçlanmaktadır. Günümüzde çoğu büyük şirketler sistemlerini daha hızlı hizmete alabilmek için monolitik yapıyla geliştirirler fakat bir süre sonra olgunlaşan projelerde kod karmaşıklığı nedeniyle sorunlar ortaya çıkabilir. Bu yapılarda esneklik ve çeviklik kaybedilebilir. Mikroservis tabanlı çözümlerde ise her servis ayrı ayrı geliştirilebildiği için ölçeklendirilebilmesi daha kolaydır. Ayrıca monolitik uygulamalarda artan işlemlerin sonrasında darboğazlar ortaya çıkabilmektedir. Bu durum verilerin ölçeklendirilmesine izin verilmediği ilişkisel veri tabanı modellerinde olabilir. Mikroservisler ve veritabanları arasındaki ön belleğe alma işlemi ile bu darboğazların önüne geçilir. Burada kullanılacak ön bellek yapısı ile işlemlerin hızı arttırılmış olur.

Sepet Tasarım Kalıbı Gömülü sistemler ve İstemci-Sunucu tasarımlı sistemlere göre daha hızlı sonuç üretebilen ve bu iki sistemin ortak avantajlarını kullanabilmek amacıyla geliştirilmiştir. Ön bellekleme için Dağıtılmış İlişkisel Veritabanı Sistemleri (DRDS) eskiden beri çok sık kullanılan bir yöntemdir. Bu yöntemde sistemin performansı diskin boyutuna ve gerçek zamanlı veri boyutuna bağlıdır. Dezavantajı ise diskin fiziksel boyutu kadar veri yüklenebileceği için depolama özelliğinin sorunu olabilir. Bunun yanında veri kaybı gibi bir güvenlik sorunu ortaya çıkabilir. Daha yeni bir yaklaşım olan Hafıza Veri Şebekesi (IMDG) ise bellek üzerinde şebekede (grid) işlem yapılan dağıtık veri yapısıdır. Sepet Tasarım Kalıbında ise IMDG genişletilebilir bellek yapısıyla ölçeklendirilebilir yapıya olanak sağlar. Güvenlik açısından ise dağıtık mimarinin güvenlik yapılarını kullanabilir.

Sepet Tasarım Kalıbının (Sidecar Pattern) geliştirilmesinde Java programlama dilinden, Spring Boot teknolojisinden, Hazelcast ön bellek teknolojisinden ve Kubernetes servisinden faydalanılmıştır. Bu teknolojileri kullanarak Gömülü, İstemci-Sunucu ve Bulut tasarım kalıplarına ek olarak ön bellek yapısının farklı şekilde yerleştirildiği esnek ve ölçeklendirilebilir bir yapı tasarlanmıştır. Farklı kullanıcı sayılarıyla, farklı sürelerle yapılan testlerde CRM uygulamasının ön bellek yapısının Sepet Tasarım Kalıbında (Sidecar Pattern) ürettiği sonuçlar diğer tasarım kalıplarının sonuçlarıyla karşılaştırılacaktır.

Bu çalışmada çok sayıda kullanıcının aynı anda ilişkisel veri tabanı modelinde tutulan kayıtlara ulaşılmak istenildiğinde yanıt verme süreleri ön bellek yapısı ile en aza indirgenmiştir. Yapılan çalışmanın, diğer yöntemlere daha az maliyetli ve servislerin her birinin kaynağını daha kolay yönetme açısından uygulanabilir bir sistem olması amaçlanmaktadır.

Anahtar Kelimeler: Mikroservis, Yazılım, Ön Bellek, Servis Durum Yöntemi, Mikroservis Mimarileri, Sidecar, Reverse Proxy

University	**:**	**T.C. İstanbul Kültür University**
Institute	**:**	**Institute of Sciences**
Department	**:**	**Computer Engineering**
Program	**:**	**Computer Engineering**
Thesis Advisor	**:**	**Assoc. Prof. Akhan AKBULUT**
Degree Awarded And Date	**:**	**MA – May 2021**

ABSTRACT

SERVICE STATUS MANAGEMENT IN THE MICROSERVIS ECOSYSTEM

In these days, the popularity of microservices is increasing. Microservices are mostly used in software, finance, banking, consulting, e-commerce and telecommunication industries. The reason for this common use is; Microservices is to make the structure flexible and scalable by using technologies together. Project that we purpose, a system will be developed to assist in the processes of constructing and placing cache structures in different design patterns of service status management in the microservis ecosystem.

The developed system, Sidecar Pattern, has intended to be used as an alternative method to Embedded, Embedded Distributed, Client-Server, Cloud design patterns used from past to present. Most large companies build their systems with a monolithic structure in order to advance faster, and problems may arise due to code complexity in projects that mature after a while. Flexibility and agility can be lost in these structures. In microservice structures, since each service can be developed separately, it is easier to scale. In these applications, bottlenecks may arise after increasing the processes. These situations can occur in relational database models that do not allow data to be scaled.

Sidecar Pattern has been developed to produce faster results than Embedded systems and Client-Server and also has been designed to used the common advantages of these two systems. Distributed Relational Database System (DRDS) is one of the common methods used from the past to the present, and in this method, the performance of the system depends on the size of the disk and the real-time data size.

The disadvantage of DRDS is that it can load as much data as the physical size of the disk, there may be a problem with the storage feature. The disadvantage of DRDS is that it can load as much data as the physical size of the disk, create a problem with the storage feature, or security problems such as data loss. On the other hand, In-Memory Data Grid (IMDG), which is a new and another approach; It is a distributed data structure processed on a grid in memory. IMDG is an approach that allows a scalable structure with an expandable memory structure and can use the security structures of the distributed architecture in terms of security. Microservice architectures are frequently used structures of a single system that are divided into multiple services that can work separately and communicate with open protocols.

In the development of Sidecar Pattern, Java programming language, Spring Boot technology, Hazelcast cache technology and Kubernetes service were utilized. Using these technologies, a flexible and scalable structure has been designed in which the cache structure is placed differently in addition to the Embedded, Client-Server and Cloud design patterns. The results of the CRM application's cache structure in the Sidecar Pattern will be compared with the results of other design patterns in the tests made with different user numbers and different durations. In this study, when a large number of users want to access the records kept in the relational database model at the same time, the response times are minimized with the cache structure.

The aim of the study is to be a system that is less costly than other methods and intended to be a system that can be applied in terms of managing the source of each of the services more easily.

1. GİRİŞ

Günümüzde mikroservismimairisnin kullanımı yaygınlaşmaktadır. Ağırlıklı olarak yazılım, finans, bankacılık, danışmanlık, e-ticaret ve telekomünikasyon sektörlerinde kullanılmaktadır. Mikroservis yapısının avantajı farklı teknolojileri bir arada kullanarak yapının esnek ve ölçeklendirilebilir hale getirilmesidir.

Dağıtılmış İlişkisel Veritabanı Sistemi (DRDS) eskiden beri çok sık kullanılan bir yöntemdi. Bu yöntemde sistemin performansı diskin boyutuna ve gerçek zamanlı veri boyutuna bağlıydı. Dezavantajı ise diskin fiziksel boyutu kadar veri yüklenebileceği için depolama özelliğinin sorunu olabilir. Bunun yanında veri kaybı gibi bir güvenlik sorunu ortaya çıkabilir. Daha yeni bir yaklaşım olan In Memory Data Grid (IMDG) ise bellek içerisinde gridde işlem yapılan dağıtık veri yapısıdır. IMDG genişletilebilir bellek yapısıyla ölçeklendirilebilir yapıya olanak sağlar. Güvenlik açısından ise dağıtık mimarinin güvenlik yapılarını kullanabilir.

Mikroservis mimarileri tek bir sistemin her biri ayrı olarak çalışabilen ve açık protokollerle iletişim sağlayabilen birden fazla servislere ayrılan yapılardır. Bu yapılar günümüzde sıkça kullanılmaktadır. Çoğu büyük şirketler sistemlerini daha hızlı ilerletebilmek için monolitik yapıyla inşaa ederler ve bir süre sonra olgunlaşan projelerde kod karmaşıklığı nedeniyle sorunlar ortaya çıkabilir. Bu yapılarda esneklik ve çeviklik kaybedilebilir. Mikroservis yapılarında ise her servis ayrı ayrı geliştirilebildiği için ölçeklendirilebilmesi daha kolaydır.

Uygulamalarda artan işlemlerin sonrasında darboğazlar ortaya çıkabilmektedir. Bu durum verilerin ölçeklendirilmesine izin verilmediği ilişkisel veri tabanı modellerinde olabilir. Mikroservisler ve veritabanları arasındaki ön belleğe alma işlemi ile bu darboğazların önüne geçilir. Burada kullanılacak ön bellek yapısı ile işlemlerin hızı arttırılmış olur.

21.yy başlarında Amazon, eBay, bestbuy.com gibi büyük şirketlerin yapıları monolitik haldeydi. Bu şirketler günümüzün getirdiği yenilikleri ve değişiklikleri sistemlerine hızlı biçimde uygulayamıyordu. Bunun nedeni ise monolitik yapının tek bir halde olması ve içerisindeki değişikliklerin sistemin tamamını etkileyecek sonuçlar doğurmasıydı. Ardışık değişiklikleri yapabilmek için bu şirketler mikroservis mimarilerini kullandılar. Sistemleri servislere ayırarak bu değişiklikleri daha hızlı adapte edebildiler. Mikroservislerin daha esnek ve ölçeklendirilebilir yapıya sahip olabilmesi için bu işlemleri sadece ilişkisel veri tabanı modellerinde yapmak yerine

uygulama ile veri tabanı arasındaki ön bellek sistemini kurgulayarak işlemlerin hızını ve verinin bütünlüğünü sağlamak amaçlanmaktadır.

Bu çalışmada çok sayıda kullanıcının aynı anda ilişkisel veri tabanı modelinde tutulan kayıtlara ulaşılmak istenildiğinde yanıt verme süreleri ön bellek yapısı ile en aza indirgenmiştir. Mikroservislerin esnek ve ölçeklendirebilir yapısı sayesinde kullanıcı ve kayıt bazlı değişimleri yaparak hızlı sonuçlar üretmek amaçlanmaktadır.

1.1. Problem Tanımı

Problemin çözümü için ilk olarak sorunun kaynağı ve sorunların kök nedenleri araştırılmaktadır. Bu kapsamda mikroservis ekosisteminde servislerin durumlarının yönetimini ve ön belleklerinin yönetimini ele alacak olursak bu yapıların Disk Veri tabanı Sistemi'nin (DDRS) yetersiz kalmasından dolayı ortaya çıkarılan yeni bir yaklaşım olduğudur. In Memory Data Grid teknolojisini kullanarak oluşan bu problemlerin çözümü hedeflenmektedir. Hazelcast teknolojisiyle veri tabanından yapılan işlemlerin ön bellekten yapılabilmesi gerekmektedir. Kullanıcıların sayısına, sistemin yoğunluğuna ve çeşitli teknolojilerin yapısına göre bu organizasyon ayarlanabilir hale gelmektedir. Farklı tasarım kalıbı implementasyonuyla örnek bir CRM uygulamasının kullanılabildiği yapılar sunulmaktadır. Mikroservis mimarileri güvenlik ve optimizasyon açısından disk tabanlı veri tabanı sistemlerine yönelik daha gelişmiş bir yapı sunmaktadır. Esnek yapısı sayesinde tek bir teknoloji kullanma sorunu ve o teknolojiye bağımlı olma durumu ortadan kaldırılacaktır.

1.2. Literatüre Katkıları

Önerilen yaklaşımlarda kullanılmak istenilen farklı tasarım kalıpları için karşılaştırmalar, implementasyonlar ve organizasyonlar sayesinde mikroservis ekosistemindeki ön bellek mimarilerinin avantajları ve dezavantajları sunulmuştur.

Tez kapsamında yapılan çalışmalarda farklı kullanıcı sayılarına göre, farklı paternlere göre, farklı metotlara göre karşılaştırma yapılmıştır. Kullanıcıya oluşturmak istediği sistem için araştırmalar ve testler sonucu bir ön bilgi verilmiştir.

Sistemin özelliklerinin Hazelcast Management Center adlı bir yönetim panelinden kullanım istatistiklerine göre değiştirilebilir olduğu, kullanıcı sayılarının istatistiklerine göre ayarlanabilir olduğu gösterilmiştir.

1.3. Tezin Organizasyonu

Bu tez 7 bölümden oluşmaktadır. Birinci bölümde, problem tanımı yapılmış, gerçekleştirilen çalışma tanıtılmış, amacı ve önemi anlatılmış ve literatüre katkısından söz edilmiştir. İkinci bölümde, tez çalışmasının konusu olan mikroservis ekosistemindeki ön bellek yapılarından, mikroservis mimarilerinden ve yapılan önceki çalışmalardan bahsedilmiştir. Tezin üçüncü bölümünde, gereksinim analizi, implementasyonlardan ve sistemler geliştirilirken kullanılan teknolojiler bahsedilmiştir. Dördüncü bölümde, önermiş olduğumuz Gömülü Tasarım Kalıbı ve Sepet Tasarım Kalıbından bahsedilmiştir. Beşinci bölüm tartışma olarak ele alınmış, tasarım kalıpları yorumlanmıştır. Altıncı bölümde bulgulardan bahsedilmiştir, test sonuçları ortaya çıkarılmıştır. Yedinci bölümde ise, yapılan çalışmanın ürettiği çıktılar üzerinden sonuçlar ele alınmıştır.

2. ÖN BİLGİLER VE LİTERATÜR ARAŞTIRMASI

Bu bölümde araştırma konusu çalışmanın temel tanımları ve literatür taraması sunulmuştur. İlk olarak mikroservisler üzerinde durulmuş ve hangi durumlarda ön bellek mimarilerinin kullanılabileceğinden bahsedilmiştir. İkinci kısımda mikroservis ekosisteminden bahsedilmiştir. Üçüncü kısımda ise ön bellek yapısı ve çeşitleri anlatılmıştır. Dördüncü bölümde ise En Son Kullanılan Ön Bellek Yöntemi (Least Recently Used) yapısı anlatılmıştır. Beşinci bölümde ise ön bellek modelleri özetlenmiştir. Son olarak mikroservis ekosistemindeki ön belleklerle ilgili yapılan çalışmalardan bahsedilmiştir.

2.1. Ön Bilgiler

Bu bölümde mikroservis yapısından, ekosisteminden ve ön bellek yapılarından bahsedilecektir.

2.1.1. Mikroservis Nedir?

Mikroservis küçük bağımsız ve modüler servisler geliştirmek için avantajlar sağlayan bir mimari yapıdır [17]. Mikroservis mimarilerde uygulamaları kendi veritabanlarına bölerek bu şekilde çalışmaları sağlanır. Servislerin birbirinden bağımsız şekilde geliştirilebilmesi ve değiştirilebilmesi mikroservislerin avantajını ortaya çıkarır.

2.1.2. Mikroservis Ekosistemi

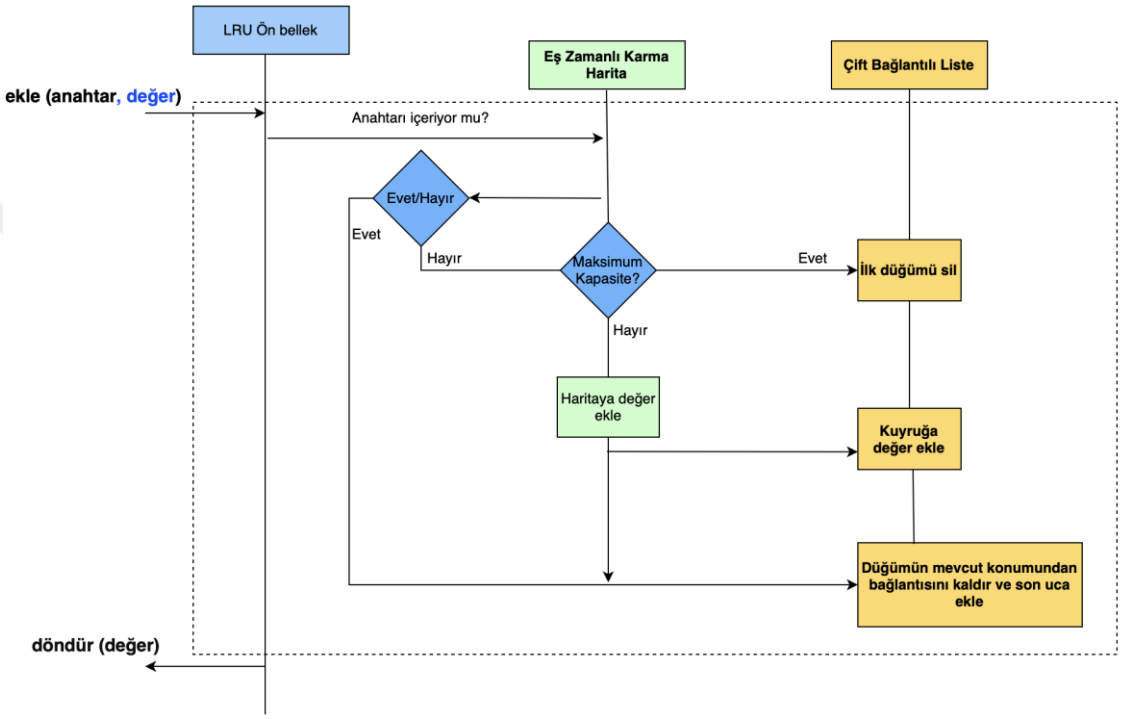
Mikroservis ekosistemi genellikle monolitik yapılardan mikroservis mimarilerine geçişler üzerinde oluşmaktadır. Farklı mimari yapılar üzerinden bu değişimler gerçekleşmektedir. Yüksek ölçeklenebilirlik yapısı, kolay ve hızlı ürünlerin oluşması ve yönetiminin parçalara ayrılmasından dolayı [1] kolay olması bu mimarilerin en önemli özelliklerindendir. Konteyner yapısı mikroservis kavramıyla daha çok popülaritesini arttırmıştır. Ekosistemin en fazla kullanıldığı sektörler; yazılım, finans ve bankacılık, danışmanlık şirketleri, e-ticaret firmaları, telekomünikasyon'dur. 2020 yılında yapılan O'REILLY firmasının araştırmasına göre bu sektörlerdeki katılımcılar 1 ile 3 yıldır mikroservisleri kullandıklarını söylemektedir [15].

2.1.3. Mikroservis Mimarisinde Ön Bellek Yapıları

Ön bellek yapısı performansı arttırmak için, back-end taraftaki yükün azaltılması için ve gecikmelerin en aza indirgenmesi [24] için kullanılan sistemlerdir. Ön belleğe alma işlemlerinin farklı yöntemleri vardır. Bu yöntemlerin dışında ön belleklerin sistem mimarilerinde nereye entegre edileceği de performansa yönelik etki etmektedir. Mikroservis mimarilerinde ön bellek yapısı her servisin içerisine veya ayrı bir sunucuya yerleştirilebilir. Bu servislere ait konteyner yapısına da eklenebilir. Ön bellek aktarım politikaları incelendiğinde First-In-First-Out (FIFO), Last-In-First-Out (LIFO), Least-Recently-Used (LRU) ve Most-Recently-Used (MRU) yapıları karşımıza çıkmaktadır [3]. Ön belleklerde iki önemli parametre vardır. Birincisi gecikme süresi(latency), ikincisi ise verimlilik oranı (hit rate) [18]. Verimlilik oranını maksimize etmek ön belleğin performansının artmasını sağlar. Bu çalışmada Least-Recently-Used (LRU) yapısı incelenmiştir ve kullanılmıştır.

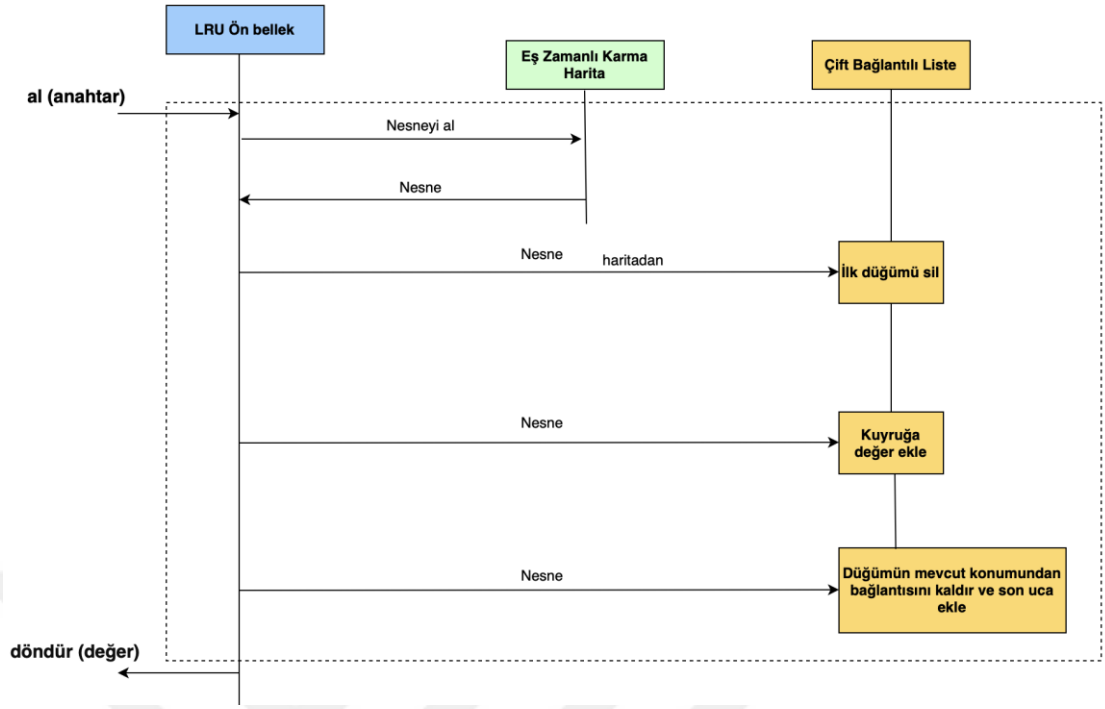
2.1.4. En Son Kullanılan Ön Bellek Yöntemi (LRU)

LRU, bir önbellek aktarım stratejisidir. Burada önbellek boyutu tahsis edilen maksimum kapasiteye ulaştığında, önbellekteki en son erişilen obje çıkarılır. Ön bellekteki objelere bir uygulamadaki birden çok iş parçacığı (thread) erişebilir. Minimum bellek kullanılması gereken çalışmalarda (basit Java uygulamaları, mobil uygulamalar veya üçüncü parti yazılımlar) bu ön bellek yapısı [24] kullanılabilir.



Şekil 2.1 - Ön belleğe obje ekleme işlemi

LRU tabanlı ön bellek yapısı iki bölümden oluşur. Birinci bölümde Eş zamanlı karma haritası (ConcurrentHashMap) işlemleri yapılır. İşlemler sırasıyla alınabilmesi için minimum bir gecikme süresine sahip olmalıdır. Eşzamanlı okuma/yazma işlemini [20] destekler. Ön belleğin limiti sabit bir boyuta ayarlanmalıdır [26]. Ön belleğe yeni bir obje eklemek için benzersiz bir anahtar kullanılmalıdır. İkinci bölümde ise basit bir Çift Bağlantılı Liste (Doubly Linked List) yapısı vardır. Bu yapıda objeyi ön bellekten çıkarmak için uygulanması gereken politika önceden belirlenmelidir.



Şekil 2.2 - Ön Bellekten obje çağırma işlemi

İkinci bölümde ise istenilen obje ön bellekten getirilirken Eş zamanlı karma haritası (Concurrent HashMap) yapısında sırayla objeler çağrılır. Objelere anahtar kullanılarak erişilebilir. Çift Bağlantılı Liste (Doubly Linked List) yapısında ise objeler bir haritada (map)'te tutulur [26]. Sıklıkla erişilen objeler en sonda, az erişilen objeler listenin başında olur. Bu politikaya göre bir LRU yöntemi belirlenmiş olur. Ön belleğe yeni bir obje eklendiğinde ve ön bellek maksimum kapasitesine ulaştığında LRU aktarımı tetiklenir. En son kullanılan obje tutulduğu map'ten silinecektir. LRU aktarım politikası farklı modellere entegre edilebilir. Bu modellerin katmanları arasında veya katmanların içerisinde ön bellek yapısının parametrelerinden birisi olarak verilir.

Ön bellek yaşam süresi (TTL): Ön belleklerin yaşam süresi, içerik tahliyecisinin bir zamanlayıcı kullanarak sona ermesi uzun süredir [2] kullanılmaktadır. Kapasite odaklı bir yaklaşımdır. İki tür TTL mevcuttur. Non-reset TLL Cache: Sıfırlanmaz. Reset TTL Cache: İçerik talep edildiğinde sıfırlanır.

2.1.5. Dağıtılmış Ön Bellek

Dağıtılmış ön bellek mimarisi, verilere erişmek için ön belleği kullanarak birden çok ağa bağlanabilen bilgisayarların tek bir sistemden yararlanmasını sağlar [22]. Diğer ön bellek mimarilerinde tek bir fiziksel sunucu varken, dağıtılmış ön belleklerde daha büyük kapasiteye ve arttırılmış işlemci gücüne sahip [23] birden çok bilgisayara sahip bir yapı [4] vardır. Bir uygulama mimarisinde birden çok yapı kullanabilir. Dağıtılmış ön belleğin avantajları şunlardır.

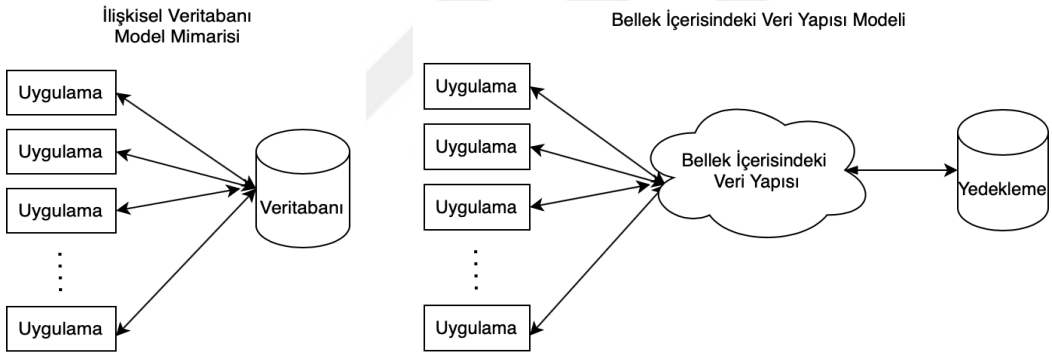
i. Uygulama hızlandırma: Disk tabanlı veri tabanı sistemleri günümüzde performans sorunlarına neden olmaktadır. En sık erişilen verileri ön bellekte depolayarak disk tabanlı veritabanlarında oluşabilecek darboğazlar azaltılabilir [8].
ii. Web oturumu bilgilerini saklama: Kullanıcının oturum bilgileri ön bellekte tutulabilir. Dağıtılmış ön bellek mimarisiyle, uygulama sunucularından herhangi biri tarafından erişilebilen çok sayıda eş zamanlı web oturumuna sahip olunur. Birden fazla uygulama sunucusu üzerinden web trafiğinin yönetilmesine olanak sağlar [8].
iii. Ağ kullanımını azaltma: Kaynak olarak veri tabanı kullanılmadan, talepler ön bellekten karşılanabilir [8]. Bu durumda sistem yüksek kullanılabilirlik (high availability) durumda kalacaktır.

Dağıtılmış mimari yapısı ön belleğin In-Memory Data Grid özelliğini kullanabilir. Bu yapının diğer sistemlere göre avantajları ve dezavantajları vardır. İlişkisel Veri tabanı Modeli Sistemi (RDBMS) ve Hafıza Veri Şebekesi (IMDG) Sistemi iki farklı sistemdir. RDBMS geçmişten günümüze sıklıkla kullanılmaktadır. IMDG ise son beş yıldır önerilen bir yaklaşımdır.

Multicast'i aktif etmek için <multicast enabled="true"> ifadesini yazmak gerekir. Bununla birlikte gelen diğer parametreler şu şekilde ayarlanabilir [9]:

<multicast-group>224.2.2.3</multicast-group>
<multicast-port>54327</multicast-port>
<multicast-time-to-live>32</multicast-time-to-live>
<multicast-timeout-seconds>2</multicast-timeout-seconds>

Parametreler arasındaki multicast-timeout-seconds değeri member'ın kendisini ana lider member etmeden önce multicast yanıtı için beklmesi gereken süreyi saniye cinsinden belirtir. Bu parametrenin çok düşük bir değer olarak ayarlanması sonucu member'lar kendi kümelerini oluşturabilirler. Bir diğer parametre olan grup ise kullanılan multicast grubun string değeridir. Port parametresi ise multicast port değerini belirlemek için kullanılır [9].



(a) (b)

Şekil 2.3 - İlişkisel Veri Tabanı Modeli Sistemi – RDBMS (a) ve Hafıza Veri Şebekesi Sistemi – IMDG (b)

Şekil 2.3'te a şemasındaki yapıda RDBMS'in performansı diske dayalı, gerçek zamanlı veritabanlarına uygundur. Dezavantajı ise diskin fiziksel boyutu kadar veri yüklenebileceği için depolama özelliğinin sorunu olabilir. Bunun yanında veri kaybı gibi bir güvenlik sorunu ortaya çıkabilir. Şekil 2.3'te b şemasındaki yapıda, IMDG genişletilebilir bellek yapısıyla ölçeklendirilebilir yapıya olanak sağlar. Güvenlik açısından ise dağıtık mimarinin güvenlik yapılarını kullanabilir. Literatürdeki araştırmalarda farklı şekilde kullanımları ve varyasyonlarının olduğu ortaya çıkmıştır. Bu yapının içerisinde dağıtılmış ön bellek yapıları da kullanılabilir.

2.2. Literatür Araştırması

Literatürleri incelediğimizde veri trafiğinin artmasındaki temel nedenlerden biri video içeriğinin hücresel ağdan akışıdır. Daha fazla baz istasyonu veya geleneksel yöntemler bu ağ trafiğini yönetmek için yeterli değildir. Önbelleğe alma işlemi mevcut ve gelecekteki internet mimarilerindeki önerilerden birisi olarak kabul edilmektedir [16].

Mikroservisler, son yıllarda işlevsellik sağlayan yazılım sistemlerine iyi tanımlanmış bir API setine sahip dizidir [25]. Mikroservis mimarisi iş mantığını birincil olarak kabul eder. Geliştirilmesi ve uygulamanın yayınlanması daha hızlı olur, çünkü her servis birbirinden bağımsız geliştirilir ve sık sık güncellenir. Bu da monolitik uygulamalar göre süreçlerin baştan başa yönetilmediği bir durumu oluşturur.

İnternette ağ trafiğini izlemenin ana parçaları algılama, kontrol ve yönetimdir. Ağ trafiği ölçümü ile karşı karşıya kalındığında veri depolama ve işlem hızı gibi sorunlar ortaya çıkar. Bu gibi durumlarda yüksek hızlı ağlarda verinin büyüklüğünü tanımlayabilmek çok önemlidir. LRU ile CBF (The Counting Bloom Filter) algoritmaları birleştirilerek bir çözüm [20] sunulmuştur. CBF algoritmasındaki V vektörünü k parçalarına bölüp hash fonksiyonları oluşturularak algoritma geliştirilmiştir. Daha sonra LRU ile CBF algoritmaları ağ trafiğini ölçmek için kullanılmıştır. Bu yeni yaklaşıma LRU_DCBF [20] adı verilmiştir.

Önbelleğe alma stratejileri büyük bant genişliği ihtiyacı duyulan video gibi öğelerin dağıtılmasında etkili bir mekanizma olarak kullanılabilir. Temelde önbelleğe alma politikaları depolama kapasitelerinden yararlanır. Etkili önbelleğe alma stratejilerinin tasarımları talep, akıllı içerik yerleştirme (intelligent content placement) ve optimum boyutlandırılacak olan önbellek sayısına [19] bağlıdır.

En Son Kullanılan Ön Bellek Yöntemi (LRU) önbelleğe alma politikalarından birisidir. Bu modelde ilk olarak en az kullanılan öğeler bellekten çıkarılır. Burada önbellek boyutu tahsis edilen maksimum kapasiteye ulaştığında, önbellekteki en az erişilen obje çıkarılır [24]. Bu çalışmada önbelleğe verilerin depolanmasından CacheMap sorumludur. Bir istemci http gönderdiğinde anahtara karşılık gelen web nesnesini alma isteğini istenen anahtar ile kontrol eder. Anahtar CacheMap'teki değeri bulur ve isteğe yanıt verir.

Bu çalışmada hesaplamalı önbelleğe alma servisleri ile ilgili olarak CoEVP proxy uygulaması kullanılmıştır. Alternatif olarak paralel hesaplamalı önbelleğe alma

servislerine ek olarak servisi kullanan kullanıcılar tarafından dinamik olarak yönetilebilen bir yapı [10] önermektedir.

Önbellek uzun sürebilecek veri işlemlerini hesaplamak üzere kullanılabilecek yazılım bileşenidir. Backend (arka uç) tarafta gecikmeleri azaltmak ve ek performans elde etmek için önbellek yapısı kullanılır. Bu çalışmada [21] Hazelcast teknolojisi kullanılmıştır ve bu teknoloji ön bellek işlemleri için iki parametre kullanılmıştır. Bunlar eşzamanlı kullanıcı sayısı ve işlenmiş verilerin boyutudur. Çalışmanın sonucunda bu parametrelerden veri boyutunun önemi eşzamanlı kullanıcı sayısının önemine göre daha fazla olmuştur. Bu testler sql sorgularıyla yapılmıştır.

Bu çalışmada mikroservis mimarilerinde onlarca kullanıcıyla gerçekleştirilen mikroservis sayısına, sunucu büyüklüklerine, iletişimin ortalama sürelerine bağlı testler gerçekleştirilmiştir [7]. Daha fazla bulut sistemi ve IoT sistemi kullanılması önerilmiştir.

Monolitik yapılarda birçok şeyi birlikte güncelleyebilirsiniz, mikroservislerde ise birçok kaynak gerekebilir. Geliştiricilerin tutarlı bir şekilde dağıtık yapıyı güncellemeleri gerekir ve bunun için de her şeyin senkronize olup olmadığını kontrol etmelidir [6]. Monolitik yapılar bu sorunlardan muaf değildir, sistemler büyüdükçe performansı arttırmak için önbelleğe alma ihtiyacı duyarlar. Bu çalışmada ağ içinde önbelleğe alma işlemi ek yükü azaltmak için verinin birer kopyası alınarak oluşturulmuştur [11]. Dağıtılmış önbelleğe sahip bir melez yapı önerilerek bir model tasarlanmıştır. Önerilen modelde, içerik mağazasındaki içeriklerin en popüler içeriği aşağıya doğru iterek ağın sınırını ve ağ popüleritesini korumak amacıyla bir yaklaşım uygulanmıştır. Ulaşılan sonuçlarda bu modelin daha iyi bir sonuç verdiği ortaya çıkmıştır [4].

3. YÖNTEM

Önerdiğimiz sistem, altı farklı modelde ön bellek yapısının ilgili modele göre kullanılması ve performans olarak karşılaştırılması sonucu önerilen yapıdır. Sistem genel hatlarıyla üç ana bileşenden oluşmaktadır. Bunlardan birincisi uygulama katmanıdır. Uygulama katmanı Spring Boot [12] teknolojisi kullanılarak oluşturulmuştur. Bu bileşenin görevi, gelen isteği alarak veri tabanı bağlantı bilgilerinin oluşturduğu iskeleti kullanarak son kullanıcıya yanıt dönmesidir. Uygulama katmanında ön bellek yapısı da kullanılmıştır. Ön belleğin boyutu, yaşam süresi ve aktarım politikası parametreleri uygulamada hazırlanmıştır. İkinci sistem ise ön bellekten okuma işlemi yapılabilmesi için açık kaynak kodlu Hazelcast [8] platformudur. Bu platform verilerin ön bellekten okunmasına, verilerin izlenmesine ve performansa yönelik değişiklikler yapılmasına olanak sağlamaktadır. Üçüncü sistem bileşeni verilerin tutulduğu ilişkisel veri tabanı modelidir. MySQL teknolojisi kullanılmıştır. Bu başlık içerisinde gereksinim analizleri, kullanılan teknolojiler ve geliştirme platformlarından bahsedilecektir.

3.1. Gereksinim Analizi

Gereksinim analizinde projenin amacı ve kapsamına yer verilmiştir. Başarılı bir yazılımın geliştirilmesi için önemli dökümanlardan bir tanesidir. Bu başlık altında sistem özellikleri, fonksiyonel olmayan sistem gereksinimleri ve kullanılan ara yüzlerden bahsedilecektir.

3.1.1. Sistem Gereksinimleri

Sistemin sorunsuz bir şekilde çalışabilmesi için aşağıdaki bahsedilen sistemin gereksinimlerini yerine getiriyor olması gerekmektedir.

3.1.1.1. Çalışma Ortamı Gereksinimleri

Çalışma ortamı için aşağıdaki gereksinimlere ihtiyaç vardır. Bu gereksinimler işletim sistemine bağlı olarak farklı sonuçlara yol açabilir.

GR3.1.1.1.A. Bilgisayar işlemci i3 veya üstü olmalıdır.

GR3.1.1.1.B. RAM miktarı minimum 4GB olmalıdır.

GR3.1.1.1.C. Sunucu için en az 8 çekirdek olmalıdır.

3.1.1.2. Yazılım Gereksinimleri

Yazılım gereksinimlerinde işletim sistemlerine bağlı olarak farklı versiyonlar ve farklı uzantıdaki programla indirilebilir. Bu uygulamaların kullanılan ilgili sürümleri aşağıda belirtilmiştir.

GR3.1.1.2.A. Sayısal ve cevap verme sürelerinin sonuçları için otomatik olarak excel dosyasına yazılmalıdır.
GR3.1.1.2.B. MySQL veri tabanı olmalıdır.
GR3.1.1.2.C. Java SE 8 sürümü olmalıdır.
GR3.1.1.2.D. Spring Boot Çerçevesinin (Framework) Maven plugin'i olmalıdır.
GR3.1.1.2.E. Hazelcast IMDG 4.2 versiyonu veya üstü olmalıdır.
GR3.1.1.2.F. Apache JMeter 3.2 veya üstü olmalıdır.
GR3.1.1.2.G. İşletim sistemi MacOS veya Windows 10 olmalıdır.

3.2. Kullanılan Teknolojiler

Bu bölümde, sistem geliştirmesi için kullanılan yazılım teknolojilerinden bahsedilmiştir.

3.2.1. Geliştirme Platformları

Geliştirilen mikroservis mimarileri ve ön bellek yapıları altı farklı model üzerinde kullanılmıştır. Bu modellerin geliştirilmesinde birden fazla yazılım dili, ön bellek teknolojisi ve veri tabanı kullanılmıştır. Bu başlık altında geliştirilen platformlardan bahsedilecektir.

- **JAVA:** Açık kaynak kodlu, nesneye yönelik ve platformdan bağımsız yapısal bir dildir. Sistemimizde Gömülü Tasarım Kalıbı (Embedded Pattern), Gömüü Dağıtılmış Tasarım Kalıbı (Embedded Distributed Pattern), İstemci-Sunucu Tasarım Kalıbı (Client-Server Pattern) ve Bulut Tasarım Kalıbı (Cloud Pattern) modelleri Java diliyle geliştirilmiştir.
- **PYTHON:** Modüler ve etkileşimli yüksek seviyeli bir programlama dilidir. Sistemimizde Sepet Tasarım Kalıbı (Sidecar Pattern) ve Ters Vekil Sepet Tasarım Kalıbı (Reverse Proxy Pattern) modelleri Python diliyle gerçekleştirilmiştir.
- **SPRING BOOT:** Hazır kodları yazmak yerine ihtiyaç olan kodların yazılmasına olanak sağlayan bir çerçeve (framework). Web sunucusu olarak Tomcat'i kullanır. Sistemimizde Gömülü Tasarım Kalıbı, Gömülü Dağıtılmış

Tasarım Kalıbı, İstemci-Sunucu Tasarım Kalıbı ve Bulut Tasarım Kalıbında bu çerçeve (framework) kullanılmıştır. Spring Boot'un sağladığı en büyük avantajlardan birisi xml konfigürasyon işlemlerini otomatik olarak yapmasıdır.

- **HAZELCAST:** Açık kaynak kodlu, Java tabanlı verileri kümelemek ve veri dağıtımını yapabilmek için tasarlanmış bir yapıdır. Verileri anlık olarak hafızada saklar. Java Sanal Makinesi (Java Virtual Machine) sayesinde verileri eşit olarak balans eder. Queue, List, Map, Set gibi birçok farklı veri tipleriyle işlem yapılmasına olanak sağlar. Sistemimizdeki tüm pattern'lerde Hazelcast teknolojisi kullanılmıştır. Maven kullanarak sistemimizdeki modellere dahil edilmiştir. Hazelcast teknolojisi Multicast adı verilen bir özelliğe sahiptir. Bu özellik sayesinde Hazelcast üyeleri (members) birbirleriyle çok noktadan iletişim kurabilir. Üyelerin birbirlerinin fiziksel adreslerine bilmesine gerek yoktur, birden fazla noktadan yayın yapılacağı için iletişim sağlanacaktır. Parametreler arasında zaman aşımı süresi ve yanıt verme süresi önemlidir. Port parametresi ile birden çok noktaya yayın yapabilmesine fayda sağlar.
- **MYSQL:** Çok iş parcacıklı, çok kullanıcılı ve hızlı çalışabilen bir ilişkisel veri tabanı yönetim sistemidir. Birçok programlama diliyle birlikte kullanılabilir. Sistemimizde oluşturulan tüm modellerde kullanılmıştır. Veriler MySQL veri tabanı üzerinde saklanmaktadır.
- **APACHE JMETER:** JMeter, başlangıçta Web uygulamalarının performans testi için geliştirilmiş fakat sonrasında farklı test senaryolarına olanak sağlayacak şekle getirilmiştir. Fonksiyonel, yük, performans, stres testi yapılabilir. Çalışmamızdaki modellerin test edilebilmesi için JMeter üzerinde HttpRequest işlemini içeren test senaryosu oluşturulmuştur. Jmeter testlerinde sistemimizde yer alan altı farklı model için yanıt verme süreleri, rampa süreleri, gecikme süresi, bağlantı süresi, kullanıcı sayısı parametrelerini içeren testler yapılmıştır. Jmeter %100 Java ile çalışan bir yapıdır ve Java Çalışma Ortamı (JRE), Java Geliştirme Kiti (JDK) kurulu olması gerekir.
- **KUBERNETES:** Google tarafından geliştirilmiş mevcutta konteyner haline getirilmiş uygulamaları dağıtmaya yarar. Konteyner yapısı CPU durumuna göre kontrol edilebilir. Operasyonel işlerin maliyetini azaltmak adına ölçekleme, otomatik dağıtma, geri adım yapma gibi özelliği bulunur. Bölme (pod) adı verilen kümelerde uygulama ve ön bellek yapıları saklanacaktır. Bu bölmelerin takibi terminal'den komutlar sayesinde yapılabilir.

4. ÖNERİLEN YAKLAŞIM: MİKROSERVİS YAPILARINDA ÖN BELLEK YÖNTEMİ

Bu çalışmada, mikroservis mimarilerindeki modeller ele alınarak bu yapılardaki performans artırımını sağlamaya yarayan ön bellek yapılarının entegre edilmesi ve konumlandırılması gerçekleştirilmiştir. Bu hedefe ulaşabilmek için hangi teknolojilerden faydalanıldığından bahsettikten sonra bu modellerde ön bellek yönteminin kullanımı, ön bellek yapısı ve genel yapısının işleyişi detaylandırılacaktır. Bu bölümde mikroservis mimarileri, kullanılan özellikleri ve uygulamaların genel işleyişi anlatılacaktır.

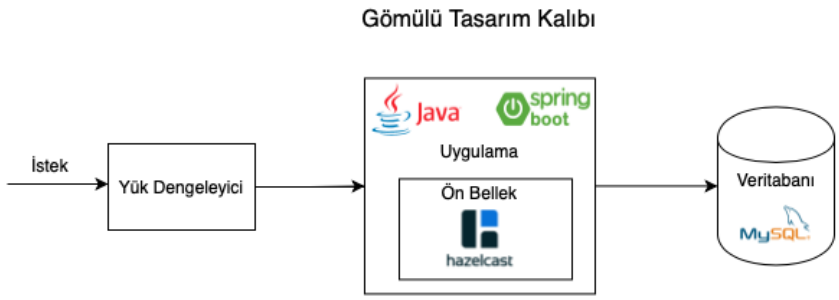
4.1. Modeller

Sistemin yapısında altı farklı model vardır. Bu modellerden Gömülü Tasarım Kalıbı, Gömülü Dağıtılmış Tasarım Kalıbı, İstemci-Sunucu Tasarım Kalıbı, Bulut Tasarım Kalıbı ve Sepet Tasarım Kalıbı implement edilmiştir. Ters Vekil Sepet Tasarım Kalıbı (Reverse Proxy Sidecar) modelinin günümüzde uygulaması henüz yoktur [12]. Geliştirilen yapılar dört ana bileşenden oluşmaktadır. Birinci bileşen isteğin alındığı Yük Dengeleyici (Load Balancer) bileşenidir. Bu bileşen sayesinde uygulamaya istek gelmeden önce iş yükünü dağıtır, isteğin kontrolünü yapar ve isteği uygulamalara yönlendirir. İkinci bileşen ise uygulamadır. Uygulama bileşeni metotların yazıldığı, veritabanı bağlantısının sınıflarının olduğu, servisleri ve modeli barındıran yapıdır. İstek alındıktan sonra üçüncü bileşen olan ön bellek yapısıyla iletişim kurar. Ön bellek yapısı her modelde farklı şekilde konfigür edilmiş ve konumlandırılmıştır. Dördüncü bileşen ilişkisel veritabanı MySQL'dir. Verilerimizin tutulduğu bu yapının uygulama ve ön bellek bileşeniyle bağlantısı sayesinde mimari modellerimizde sorgulama işlemlerinin yapılması sağlanmaktadır [12]. Bu bölümde geliştirilen modellerin yapısı, kullanılan teknolojiler ve iş akışı incelenecektir.

4.1.1. Gömülü Tasarım Kalıbı

Gömülü modelde süreli olan bir iş organizasyonu, bir hesaplama veya veritabanındaki sorgulama gibi işlemler yapılabilir. Ön belleğe alma işlemi yerleşik veri yapılarını veya bazı ön belleğe alma çerçevelerini kullanarak yapılabilir [14]. Ön bellek uygulama katmanına yerleştirilmiştir. Bu modelin çalışma mantığı sırasıyla şu şekildedir.

- İstek (Request), Yük Dengeleyici'a (Load Balancer) gelir.
- Load Balancer, isteği uygulama katmanına iletir.
- Uygulama katmanı isteği alır, isteğin ön bellekte olup olmadığını kontrol eder. Eğer istek ön bellekte varsa, değeri döndürür. Değilse, uzun süreli işi yani veritabanına giderek sorgulamayı yapar, değeri döndürür.



Şekil 4.1 – Gömülü Tasarım Kalıbı

Bu uygulamayla, ön bellek katmanına eklenen @Cacheable ifadesiyle yazılan fonksiyonda değeri döndürür. Bu işlem Şekil 4.2'te olduğu gibidir.

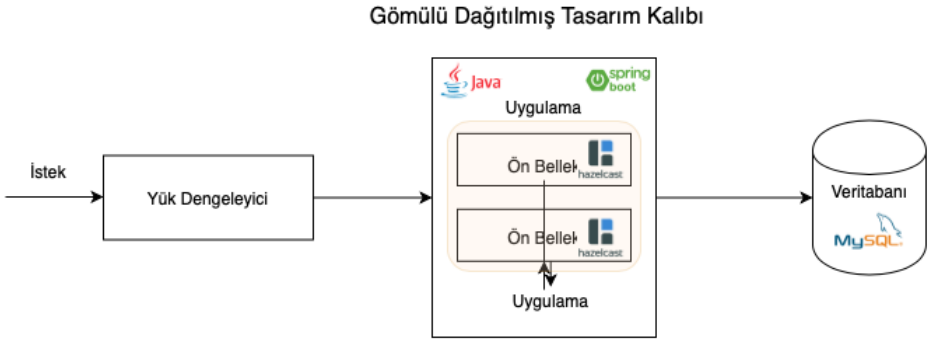
```
11  @Service
12  public class UserService {
13
14      @Autowired
15      private UserRepository repository;
16
17      @Cacheable(cacheNames = { "userCache" })
18      public List<User> getUsers() {
19          System.out.println("getAllUsers metotu ilk kez çağrılıyor.");
20          return repository.findAll();
21      }
```

Şekil 4.2 – Servis Yapısının Oluşturulması

Gömülü Tasarım Kalıbının dezavantajı ise, iki kez gelen isteği Yük Dengeleyici alır ve ayrı ayrı tanımlanmayan bir ön bellek yapısı olmadığı için bu işlemi veritabanına giderek iki kez yapmak zorunda kalır. Bu modelin geliştirilmesi gerekmektedir ve dağıtılmış bir yapı kullanılması gerekir.

4.1.2. Gömülü Dağıtılmış Tasarım Kalıbı

Gömülü dağıtılmış modelde ön belleği dağıtık bir şekilde kullanmaya yarayan Hazelcast teknolojisi kullanılmıştır. Tüm ön bellekler, tek bir ön bellek kümesi oluşturur ve uygulamalar dağıtılmış yapıdaki bu kümeyi kullanır. Hazelcast, Java Spring Boot çerçevesi (framework) ile birlikte kullanılabilir.



Şekil 4.3 – Gömülü Dağıtılmış Tasarım Kalıbı

Herhangi bir ek yapılandırma gerektirmeden, Hazelcast Instance'ı kullandık. Bu yapının sağladığı avantaj ise, Java Sanal Makinesi (Java Virtual Machine) ile aynı fiziksel makinede çalıştığı için verilerin getirilmesindeki gecikme az olmaktadır.

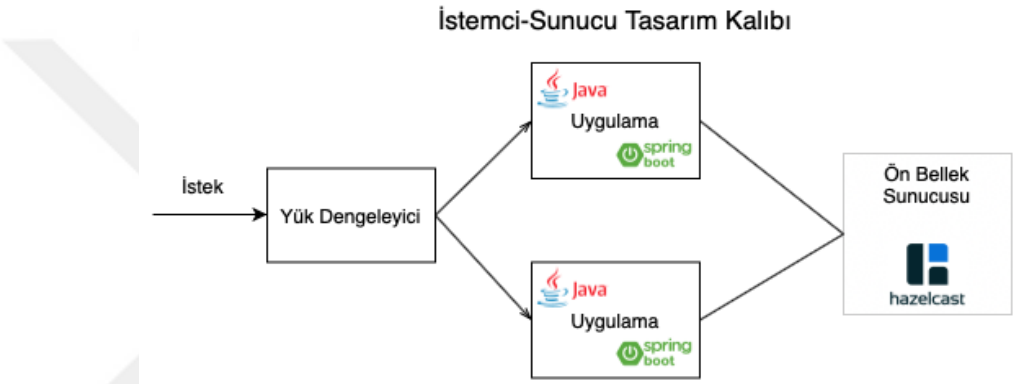
```
9   @Configuration
10  public class CacheConfiguration {
11      @Bean
12      CacheManager cacheManager()
13      {
14          return new
15                  HazelcastCacheManager(Hazelcast.newHazelcastInstance());
16      }
17  }
```

Şekil 4.4 – Dağıtılmış Servis Yapısının Oluşturulması

Şekil 4.4'teki CacheConfiguration sınıfında yapılan yapılandırma sayesinde bu yapı oluşturuldu. Bu modellere ek olarak farklı iki makinenin birbiriyle iletişimini sağlayan bir model olan İstemci-Sunucu modelinden bahsedilecektir.

4.1.3. İstemci-Sunucu Tasarım Kalıbı

İstemci-Sunucu modeli sıklıkla kullanılan veritabanı modeliyle benzerlik göstermektedir. Merkezi bir sunucu vardır. Uygulamalar bu sunucuya bağlanır. Gömülü modellerden birinci farkı, ön bellek yapısının İstemci-Sunucu Tasarım Kalıbında aynı sunucuda olmasıdır. Gömülü Tasarım Kalıbında ise ön bellek yapısını ayrı ayrı yönetebiliriz (ölçeklendirme, yedekleme, güvenlik). İkinci farkı ise, uygulamanın ön bellek kullanması için bir istemci kütüphanesi kullanması gerekir. Bu durum uygulamaların farklı programlama dili ve protokollerle yapılabileceği avantajını sağlar.



Şekil 4.5 – İstemci-Sunucu Tasarım Kalıbı

Bu yapıda Spring Boot cache adı verilen bir yapılandırma türüne ihtiyaç vardır. Bu yapı sayesinde istemci Kubernetes tarafında bir sunucu ile iletişime geçebilir. Aynı şekilde diğer sistemlerde olduğu gibi bir Hazelcast instance'ı da oluşacaktır.

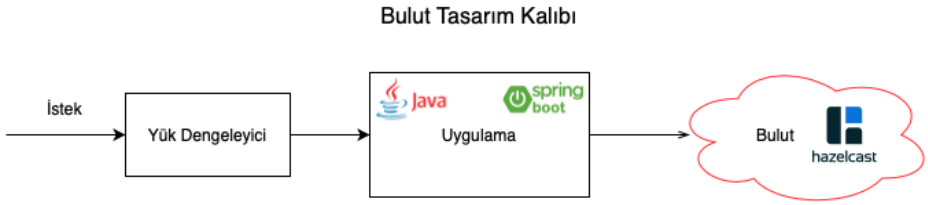
```
@Configuration
public class CacheConfiguration {

    CacheManager cacheManager() {
        ClientConfig clientConfig = new ClientConfig();
        clientConfig.getNetworkConfig().getKubernetesConfig().setEnabled(true);
        return new
                HazelcastCacheManager(
                        HazelcastClient.newHazelcastClient(clientConfig));
    }
}
```

Şekil 4.6 – Ön Bellek Yapılandırma Sınıfı

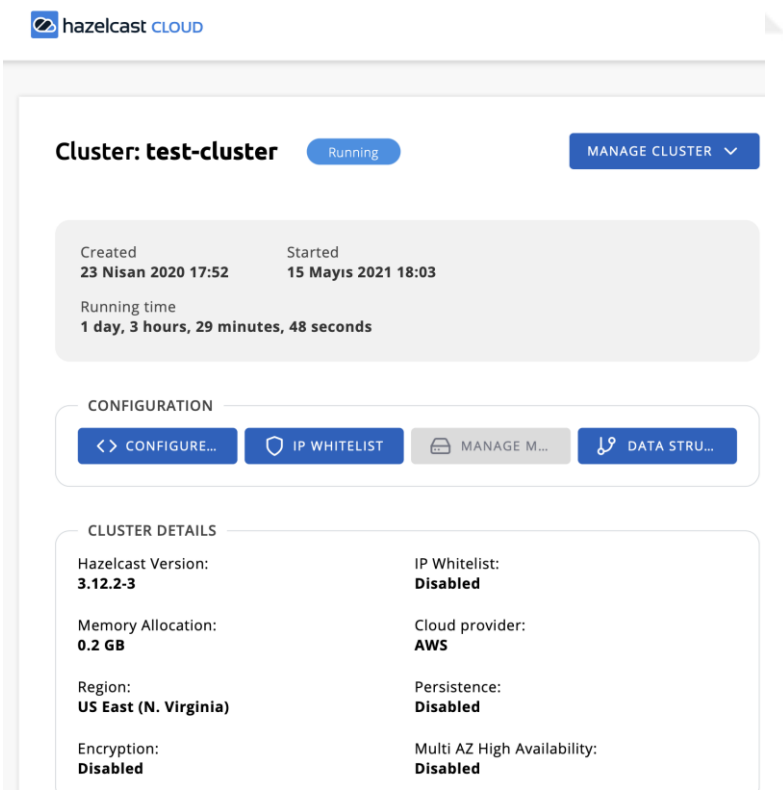
4.1.4. Bulut Tasarım Kalıbı

Mimari olarak İstemci-Sunucu Tasarım Kalıbına benzemektedir. Farkı ise sunucunun dışarıda bir bulut ortamında tutulmasıdır. Organizasyonel işlemler bulut hizmeti alınan firma tarafından sağlanır. Hazelcast instance bulut ortamında saklanmıştır.



Şekil 4.7 – Bulut Tasarım Kalıbı

Bu yapıda Hazelcast bulut özelliğini kullanabilmek için https://cloud.hazelcast.com/ adresinde bir küme oluşturulmuştur. Yönetim panelinden oluşturulan bu kümenin yönetimi, güvenliği ve takibi sağlanabilir. Performans takibi açısından kaynak arttırımı veya azaltımı yapılabilir. Maliyet olarak uygulamanın kullanılmadığı durumlarda cluster durdurulabilir.



Şekil 4.8 – Hazelcast Cloud Platformundaki Küme Bilgileri

Konfigürasyon işlemleri ön bellek verilerini ayırmak için ayrı bir yönetim (ölçeklendirme, yedekleme) programında yapılmaktadır. Bu durum herhangi bir programlama dilini kullanmaya olanak sağlar. Küme için verilen bir token kullanılarak güvenlik açısından geliştirme sağlanmış olur.

```
12  @Configuration
13  public class CacheConfiguration {
14      @Bean
15      CacheManager cacheManager() {
16          ClientConfig clientConfig = new ClientConfig();
17          clientConfig.getNetworkConfig().getCloudConfig()
18              .setEnabled(true)
19              .setDiscoveryToken("fbUcDsiVYqgi0AZwwlLoPINtzsrfXmP63LzZbvyqzcZedHZFAk");
20          clientConfig.setClusterName("test-cluster");
21          return new HazelcastCacheManager (
22              HazelcastClient.newHazelcastClient(clientConfig));
23
24      }
25  }
```

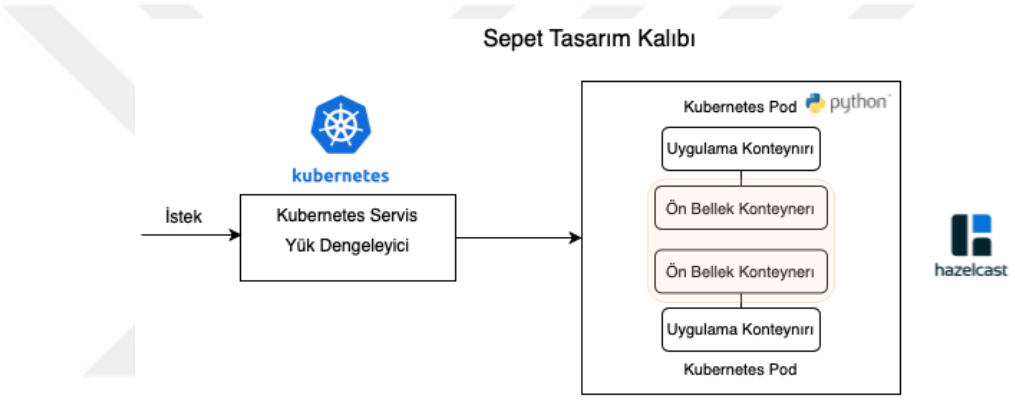
Şekil 4.9 – Bulut Konfigürasyon Sınıfı

Bu yapının getirdiği ek bir sorumluluk ise, ön bellek uygulamayla aynı sunucuda olmayacağı için ön belleğin olduğu sunucuyu yönetmek ve takip etmek gerekir. Açıklanan dört model de geçmişten günümüze sıklıkla kullanılan yapılardır. Bu yapılara ek olarak çok fazla kullanılmayan Sepet Tasarım Kalıbında ön bellek işlemleri farklılık gösterebilir.

4.1.5. Sepet Tasarım Kalıbı

Sidecar sepet düzeni olarak adlandırılabilir. Bu düzen Kubernetes ortamına özgüdür. Dağıtım birimlerine Kubernetes Pod denir. Uygulama bir bölme (pod) üzerindedir ve Python programlama diliyle geliştirilmiştir. Bölmenin içerisinde ek olarak ön bellek yapısı bulunur. Ön bellek yapısının bulunduğu bölmelere sidecar bölmeleri denmektedir. Bu modelin çalışma mantığı şu şekildedir.

- İstek, Kubernetes Yük Dengeleyici'ye (Kubernetes Load Balancer) gelir ve uygulamaların olduğu bölmelerden birine iletilir.
- Uygulama bölmelerinden ön bellek konteyner'ına gelir. Bu işlem localhost adı verilen ön bellek sunucusuna gider.



Şekil 4.10 – Sepet Tasarım Kalıbı

Sepet Tasarım Kalıbının Gömülü Tasarım Kalıbına benzeyen özellikleri uygulama ile aynı makinede olması, kaynakların uygulama ve ön bellek ile paylaşılmasıdır. İstemci-Sunucu Tasarım Kalıbına benzeyen özellikleri ise, herhangi bir programlama diliyle geliştirilebilir olması ve ön belleğin izolasyona sahip olmasıdır. Büyük ölçekli bir projeyse Sepet Tasarım Kalıbını kullanabiliriz, çünkü bu yapıda uygulamaların her birine farklı şekilde kaynak ayırıp ön bellek mimari yapımızı da uygulamalarla birlikte Kubernetes bölmelerinde (pod) saklayabiliriz. Konfigürasyon yapısı İstemci-Sunucu modelinin konfigürasyonuna benzerdir [13].

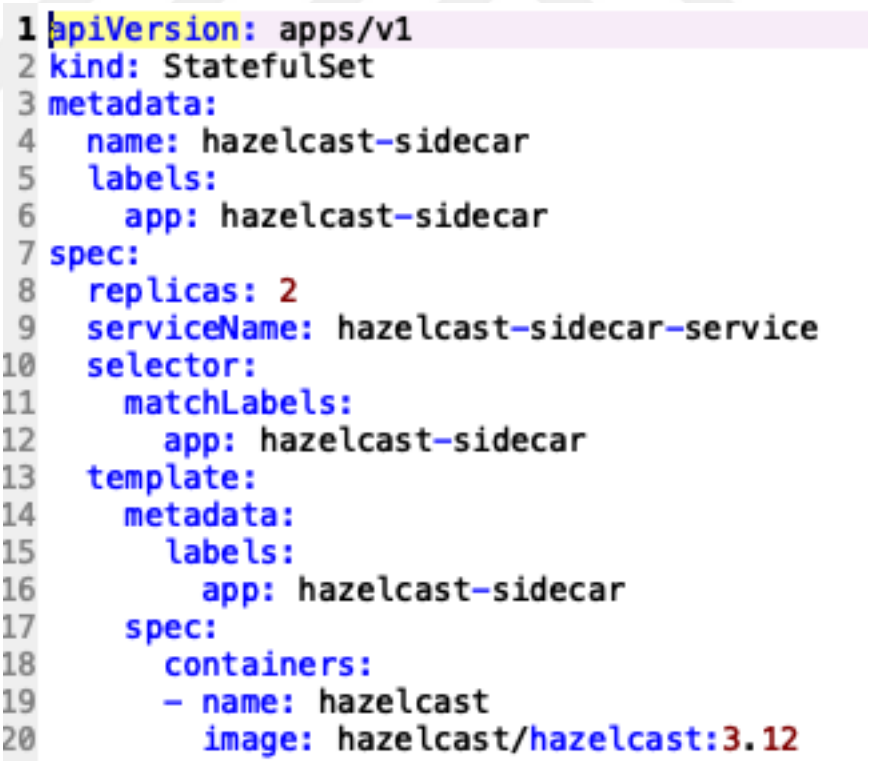
```
21 app = Flask (__name__)
22 config = hazelcast.ClientConfig();
23 config.network_config.addresses.append("localhost:5704")
24 hazelcastClient=hazelcast.HazelcastClient(config)
25
```

Şekil 4.11 – Sepet Tasarım Kalıbı Hazelcast Konfigürasyon Yapısı

Sepet Tasarım Kalıbı Gömülü ve İstemci-Sunucu modellerinin ana bir konteyner yapısında tutulmasını sağlayan bir modeldir. Projenin izleme (monitoring) bileşeni Docker bileşeninde yer alabilir. Sepet (sidecar) topolojisi, uygulamayla birlikte ölçeklenir ve her iki konteyner de ayrı makinede çalışır. İstemci kitaplıklarıyla Hazelcast üyelerine bağlanır. Yapılan çalışmalarda şu durumlarda Sepet Tasarım Kalıbının kullanılabileceği ortaya çıkmıştır.

- Java Sanal Makineleri (Java Virtual Machine) olmayan yapılarda Gömülü modelle birlikte entegre edilebilir.
- Kubernetes ile birlikte Hazelcast teknolojisinin birlikte çalışabildiği Java dışı bir dille çalışabilir durumlarda kullanılabilir.
- Sepet Tasarım Kalıbı, İstemci-Sunucu yapısı içeren bir uygulama gerekiyorsa kullanılabilir. Bu durumda Hazelcast üyeler arasında iletişimler yapılabilir.

Sepet Tasarım modelinde Python programlama dili ve web tabanlı iki metot içeren yapıdadır. Uygulama dockerize edilmiştir. Yani Dockerfile adı verilen formatta bir docker görüntüsü oluşturulmuştur. Bu modelin dağıtımı ise deployment.file dosyasındaki bilgilerle gerçekleşmiştir.



```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: hazelcast-sidecar
  labels:
    app: hazelcast-sidecar
spec:
  replicas: 2
  serviceName: hazelcast-sidecar-service
  selector:
    matchLabels:
      app: hazelcast-sidecar
  template:
    metadata:
      labels:
        app: hazelcast-sidecar
    spec:
      containers:
      - name: hazelcast
        image: hazelcast/hazelcast:3.12
```

Şekil 4.12 – Hazelcast Dağıtım Sınıfı

Uygulamanın adı, docker imajı bu dosyada belirtilmiştir. Konfigürasyon dosyasındaki hazelcast-sidecar-service adında bir servis oluşturulmuştur. Bu servislerin takibi terminal üzerinden kontrol edilmiştir. Uygulamanın tutulduğu yer bu modelde Kubernetes pod adı verilen yapıdır [13]. Bu yapıların yönetimi diğer modellere göre farklıdır. Diğer ön bellek modellerine ek olarak Sepet Tasarım Kalıbında pod yönetimi ve takibi eklenmiştir.

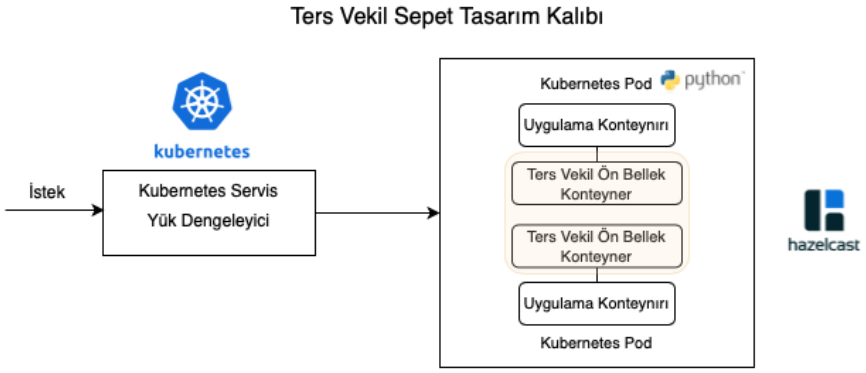
```
furkankaratas@Furkan-MacBook-Pro ~ % kubectl get pods
NAME                  READY   STATUS    RESTARTS   AGE
hazelcast-sidecar-0   2/2     Running   36         145d
hazelcast-sidecar-1   2/2     Running   37         145d
```

Şekil 4.13 – Hazelcast Pod Durum Takibi

4.1.6. Ters Vekil Sepet Tasarım Kalıbı

Ters Vekil Sepet Tasarım Kalıbı, Sepet Tasarım Kalıbının farklı bir varyasyonudur. Bu yapı sırasına göre şu şekildedir.

- İstek, Kubernetes Yük Dengeleyici'ye (Kubernetes Load Balancer) gelir ve uygulamaların olduğu bölmelerden birine iletilir.
- Ters Vekil Ön Bellek Konteyner (Cache Container) adı verilen ters vekile sahip ön bellek yapısını içeren yapı bu isteği alır.
- Ön bellek bu işlemin daha önceden alınıp alınmadığını kontrol eder.
- Eğer alındıysa, ön belleğe alınmış yanıtı döndürür.



Şekil 4.14 – Ters Vekil Sepet Tasarım Kalıbı

Bu yapının Sepet Tasarım Kalıbından farkı uygulamanın ön belleğin var olmasından bilgisinin olmamasıdır [5]. Literatür taraması yapıldığında şu an halihazırda Ters Vekil Sepet Tasarım Kalıbının (Reverse Proxy Sidecar Pattern) geliştirilmiş bir yapısı yoktur.

5. BULGULAR

Araştırmanın bu bölümünde, bulgular, bulgulara ait yorumlara dayalı olarak elde edilen sonuçlar, yapılan modellerin performans testlerinin sonuçlar ve öneriler yer almaktadır. Ön bellek yapısının mikroservis mimarilerindeki kullanımlarının, konumlandırılmasının ve uygulamalarla birlikte entegre edilmesinin sonuçlarından bahsedilecektir. Deneylerimizde farklı modelleri gerçekleştirmek amacıyla kullanıcı kimliği, adresi ve adı bilgilerinin tutulduğu bir CRM uygulama senaryosunu kullandık. Tüm bu bilgiler 10, 100, 1000 kullanıcı ile çalıştığında farklı sonuçlar ortaya çıkmıştır. Ön bellek modelleri uygulandı ve sonuçları karşılaştırıldı. Modellerin birbirleriyle kıyaslanması, performans testlerindeki farklılıkların sebepleri açıklanacaktır. Test sonuçları detaylı bir şekilde alt başlıklar halinde paylaşılmıştır.

5.1. Gömülü Tasarım Kalıbı ve Gömülü Dağıtılmış Tasarım Kalıbının Kıyaslanması

Testler Jmeter üzerinde yapılmıştır. Bu testler 10, 100, 1000 kullanıcı sayılarına göre oluşturulmuştur. Rampa süresi, kullanıcıların sisteme toplam giriş sürelerini oluşturmaktadır. 10, 100 kullanıcı için 10 saniye rampa süresi yetertli olmuştur. 1000 kullanıcı içinse 20 saniye yeterli olmuştur. Yapılan çalışmalarda iki farklı metot üzerinden kıyaslamalar yapılmıştır. CRM uygulama senaryomuzda 1000 adet kullanıcı bilgilerini içeren kayıt içeren bir tablo tutulmaktadır. Bu tablodan ilk olarak tüm kayıtları getiren ve ön bellekten okuma yapan metot mevcuttur. İkinci metot ise, bu kayıtlardan sadece bir kaydı getiren metottur. Karşılaştırmalar bu metotlar üzerinden yapılmıştır.

Test 1

Bu kıyaslamada, *getAllUsers()* metodu için test sonuçları vardır. Test için JMeter programı üzerinde ayarlamalar yapılmıştır. İş parçacığı grubu Apache JMeter üzerinde şu şekilde konfigür edilmiştir:

Thread kullanıcı sayısı: 10

Rampa süresi: 10 sn

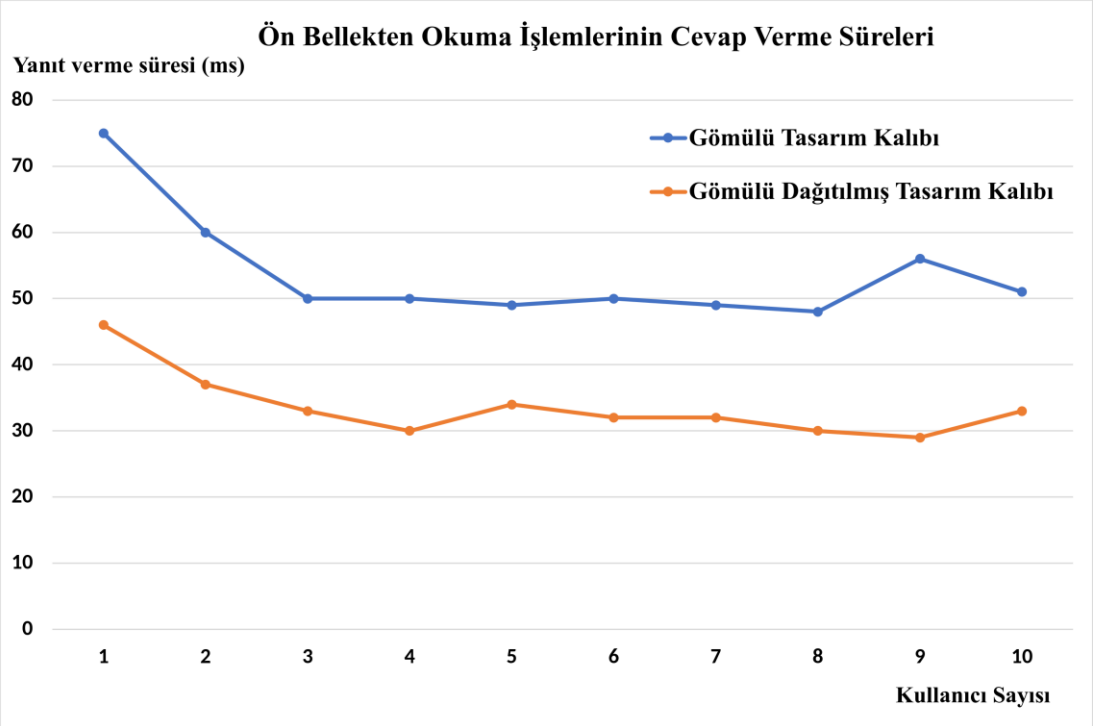
Döngü: 1

Http İsteği: http://localhost:8080/cache/getAllUsers

Test iki tasarım kalıbının aynı metot için aynı parametrelerle Apache JMeter programındaki konfigürasyonlar seçilerek gerçekleştirilmiştir. Parametrelerimiz sırasıyla ön bellek tasarım kalıbı, kullanıcı sayısı, rampa süresi ve metottur.

{Gömülü Tasarım Kalıbı, 10, 10, getAllUsers()}

{Gömülü Dağıtılmış Tasarım Kalıbı, 10, 10, getAllUsers()}



Şekil 5.1 – Gömülü Tasarım Kalıbı ve Gömülü Dağıtılmış Tasarım Kalıbı için 10 Kullanıcıyla Yapılan Tüm Tabloyu Sorgulama Sonuçları

Bu karşılaştırmada 10 iş parçacığı aynı anda uygulama üzerinden istek gönderilerek veri tabanındaki tabloda yer alan 1000 kaydı çağırmıştır. Gömülü

Tasarım Kalıbı yönteminde veriler ortalama olarak 53,8 ms süresinde gelmektedir. Gömülü Dağıtılmış Tasarım Kalıbı yönteminde ise veriler ortalama olarak 33,6 ms süresinde gelmektedir. Gömülü Tasarım Kalıbı yönteminde tek bir member kullanılır, Gömülü Dağıtılmış Tasarım Kalıbı ise iki tane farklı member cluster yapısı oluşturur ve Hazelcast'ın multicast özelliğini kullanarak verileri ön bellekten daha hızlı sürede getirmiş olur. Gömülü Dağıtılmış Tasarım Kalıbında dağıtılmış bir ön bellek yapısı bulunur, aynı anda gelen iki isteği kontrol etmez ve direkt ön belleğe yönlendirir. Bu da diğer yönteme göre performans arttırımı sağlar.

Multicast özelliği sayesæinde Hazelcast member'ların çok noktaya erişim sağlayarak birbirleriyle iletişim kurmasını sağlar. Kümede oluşan member'ların birbirlerinin somut adresini bilmelerine gerek yoktur, birden fazla noktaya yayın yapılacağı için iletişim sağlanmış olur. Multicast mekanizmasının mümkün olup olmayacağı oluşturulan ortama ve konfigür edilebilir bir yapıya bağlıdır.

Test 2

Bu kıyaslamada, *getAllUsers()* metodu için test sonuçları vardır. Test için JMeter programı üzerinde ayarlamalar yapılmıştır. İş parçacığı grubu Apache JMeter üzerinde şu şekilde konfigür edilmiştir:

Thread kullanıcı sayısı: 100

Rampa süresi: 10

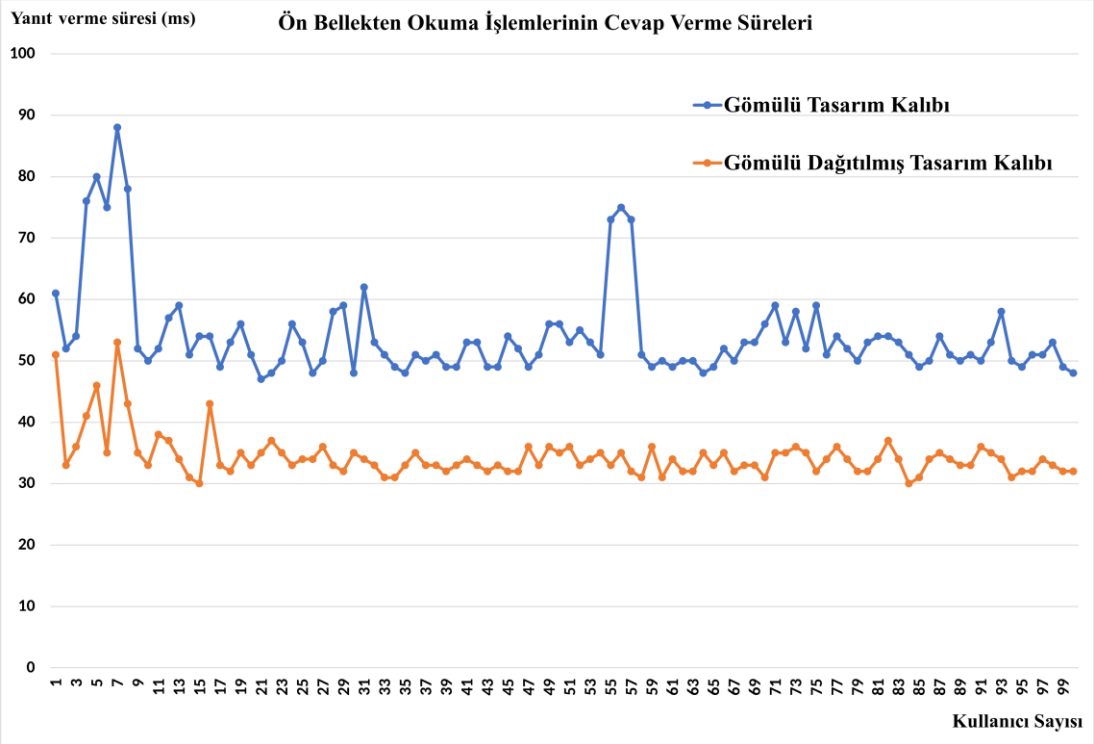
Döngü: 1

Http İsteği: http://localhost:8080/cache/getAllUsers

Test iki tasarım kalıbının aynı metot için aynı parametrelerle Apache JMeter programındaki konfigürasyonlar seçilerek gerçekleştirilmiştir. Parametrelerimiz sırasıyla ön bellek tasarım kalıbı, kullanıcı sayısı, rampa süresi ve metottur.

{Gömülü Tasarım Kalıbı, 100, 10, getAllUsers()}

{Gömülü Dağıtılmış Tasarım Kalıbı, 100, 10, getAllUsers()}



Şekil 5.2 – Gömülü Tasarım Kalıbı ve Gömülü Dağıtılmış Tasarım Kalıbı için 100 Kullanıcıyla Yapılan Tüm Tabloyu Sorgulama Sonuçları

Yapılan testte ilk 10 thread kullanıcısının çağırdığı ortalama süre Şekil 5.1'de olan ortalama süre ile aynı olduğu gözlemlenmiştir. Bu karşılaştırmada 100 iş parçacığı aynı anda uygulama üzerinden istek gönderilerek veri tabanındaki tabloda yer alan 1000 kaydı çağırmıştır. Gömülü Tasarım Kalıbı yönteminde veriler ortalama olarak 54,17 ms süresinde gelmektedir. Gömülü Dağıtılmış Tasarım Kalıbı yönteminde ise ortalama olarak 34,33 ms süresinde gelmektedir. Gömülü Tasarım Kalıbı yönteminde tek bir member kullanılır, Gömülü Dağıtılmış Tasarım Kalıbı yönteminde ise iki tane farklı member cluster yapısı oluşturur ve Hazelcast'ın multicast özelliğini kullanarak verileri ön bellekten daha hızlı sürede getirmiş olur.

Test 3

Bu kıyaslamada, *getAllUsers()* metodu için test sonuçları vardır. Test için JMeter programı üzerinde ayarlamalar yapılmıştır. İş parçacığı grubu Apache JMeter üzerinde şu şekilde konfigür edilmiştir:

Thread kullanıcı sayısı: 1000

Rampa süresi: 20

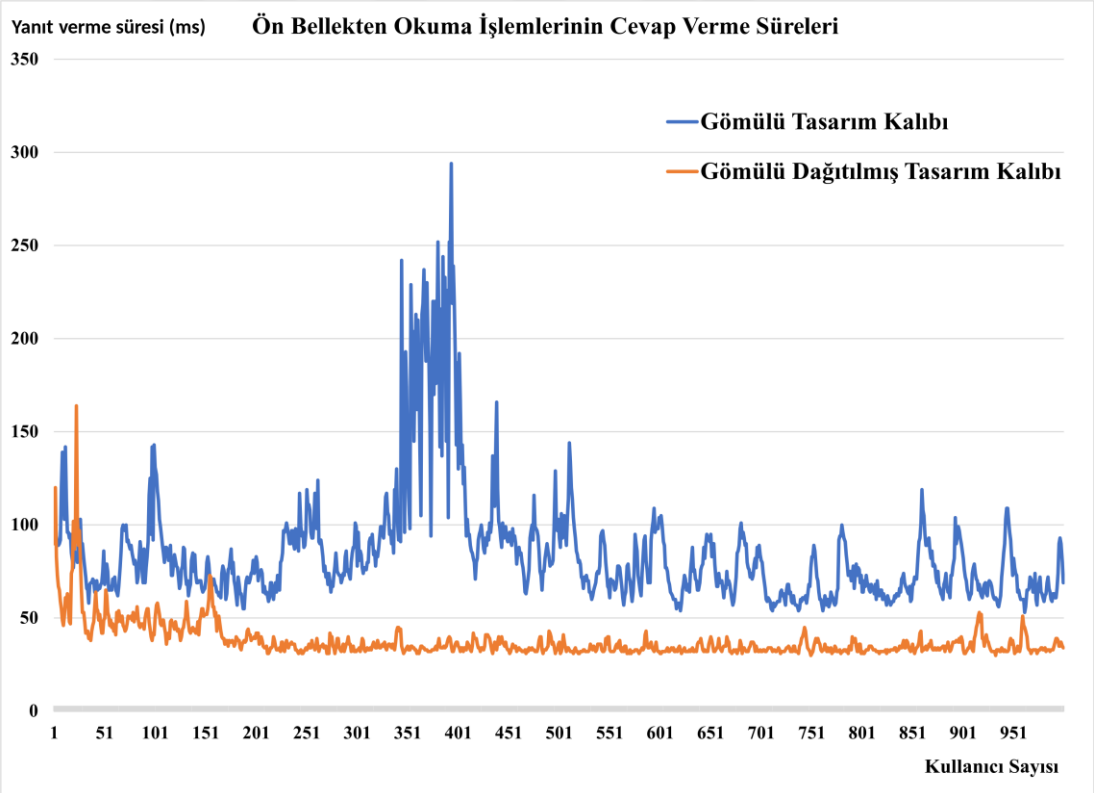
Döngü sayısı: 1

Http İsteği: http://localhost:8080/cache/getAllUsers

Test iki tasarım kalıbının aynı metot için aynı parametrelerle Apache JMeter programındaki konfigürasyonlar seçilerek gerçekleştirilmiştir. Parametrelerimiz sırasıyla ön bellek tasarım kalıbı, kullanıcı sayısı, rampa süresi ve metottur.

{Gömülü Tasarım Kalıbı, 1000, 20, getAllUsers()}

{Gömülü Dağıtılmış Tasarım Kalıbı, 1000, 20, getAllUsers()}



Şekil 5.3 – Gömülü Tasarım Kalıbı ve Gömülü Dağıtılmış Tasarım Kalıbı için 1000 Kullanıcıyla Yapılan Tüm Tabloyu Sorgulama Sonuçları

Bu karşılaştırmada 1000 iş parçacığı aynı anda uygulama üzerinden istek gönderilerek veri tabanındaki tabloda yer alan 1000 kaydı çağırmıştır. Bu iş parçacığı kullanıcıları toplam 20 saniyede alınmıştır, ilk saniyede 50, ikinci saniyenin sonunda

100 thread user oluşturulmuş ve kayıtlar çağrılmıştır. Yapılan testte ilk 100 thread kullanıcısı için sonucun Şekil 4'teki sonuçla aynı olduğu görülmüştür. Gömülü Tasarım Kalıbı Pattern yönteminde veriler ortalama olarak 85,13 ms süresinde gelmektedir. Gömülü Dağıtılmış Tasarım Kalıbı yönteminde ise ortalama olarak 37,67 ms süresinde gelmektedir. Gömülü Tasarım Kalıbı yönteminde tek bir member kullanılır, Gömülü Dağıtılmış Tasarım Kalıbı yönteminde ise iki tane farklı member cluster yapısı oluşturur ve Hazelcast'ın multicast özelliğini kullanarak verileri ön bellekten daha hızlı sürede getirmiş olur.

Gömülü Tasarım Kalıbının testi sonucunda 340-397 thread kullanıcı arası çağrılma süreleri artmıştır. Buradaki etkili olan neden gecikme sürelerinin ortalama olarak iki katına çıkmış olmasıdır. Gömülü Tasarım kalıbının tümüne bakıldığında ortalama 75ms bir gecikme süresi varken bu thread kullanıcılar arasında bu süre 150ms'ye çıkmıştır. Buradaki gecikmelerin oluşmasının nedeni ise belirtilen saniyeler arasında grup thread kullanıcıların sayısının fazla oluşmasından kaynaklıdır. Geçiş Kontrol Protokolü (Transmission Control Protocol – TCP) seviyesinde alınan isteğin uygulama sunucusuna aktarılması, isteğin sunucu üzerindeki ön bellekten okuma işlemini yaptığı süre, cevabın Geçiş Kontrol Protokolü (Transmission Control Protocol – TCP) üzerinden iletilmesi bu gecikmenin ana nedenlerindendir.

Tablo 5.1 – Gömülü Tasarım Kalıbı ve Gömülü Dağıtılmış Tasarım Kalıbı için Tüm Tabloyu Sorgulayan Karşılaştırma

Ön Bellek Tasarım Kalıpları	Kullanıcı Sayısı	Rampa Süresi	Düğüm (Node) Sayısı	Ortalama Yanıt Verme Süresi
Gömülü Tasarım	10	10	1	53,8
Gömülü Dağıtılmış	10	10	2	33,6
Gömülü Tasarım	100	10	1	54,17
Gömülü Dağıtılmış	100	10	2	34,33
Gömülü Tasarım	1000	20	1	85,13
Gömülü Dağıtılmış	1000	20	2	37,67

Bu iki modelde de ilişkisel veritabanı modelinden okumaya göre daha hızlı getirilmiştir. Gömülü Dağıtılmış Tasarım Kalıbı 10 kullanıcılı yapılan testte ortalama 20,2ms daha hızlı, 100 kullanıcılı yapılan testte ortalama 19,84ms daha hızlı, 1000 kullanıcılı yapılan testte ise ortalama 47,46ms daha hızlı yanıt vermiştir.

Test 4

Bu kıyaslamada, *getUserById()* metodu için test sonuçları vardır. Test için JMeter programı üzerinde ayarlamalar yapılmıştır. İş parçacığı grubu Apache JMeter üzerinde şu şekilde konfigür edilmiştir:

Thread kullanıcı sayısı: 10

Rampa süresi: 10

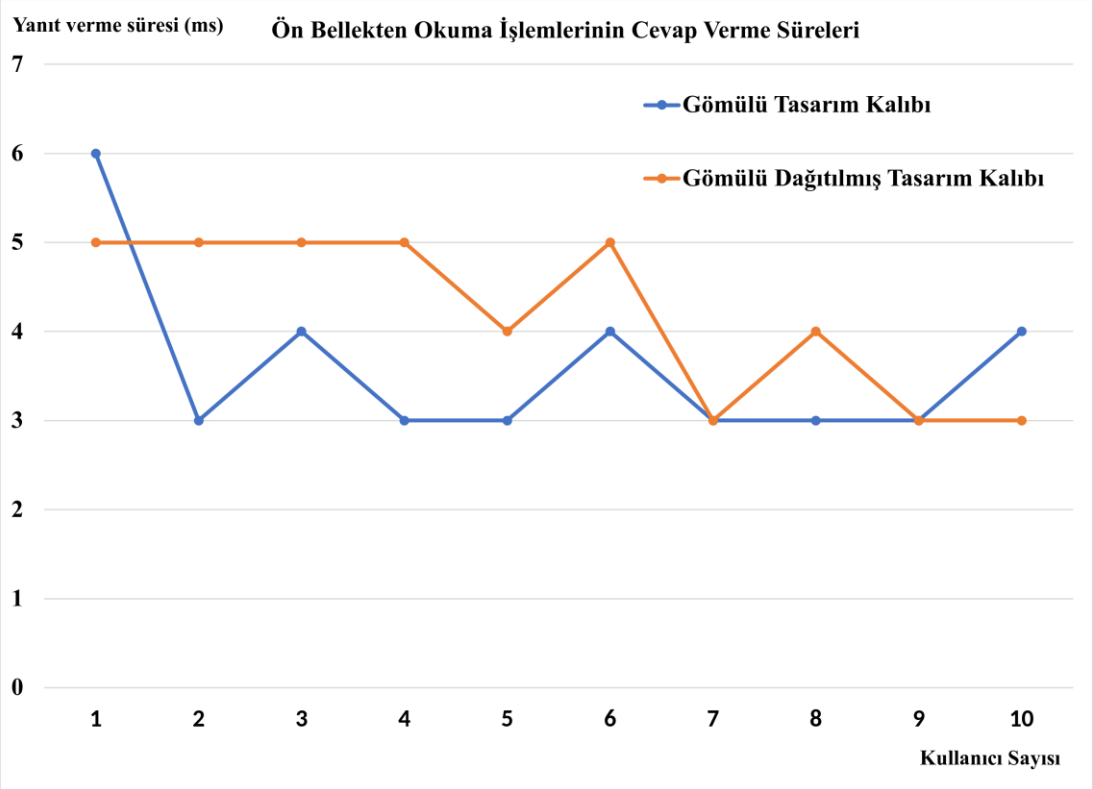
Döngü sayısı: 1

Http İsteği: http://localhost:8080/cache/getUserById/455

Test iki tasarım kalıbının aynı metot için aynı parametrelerle Apache JMeter programındaki konfigürasyonlar seçilerek gerçekleştirilmiştir. Parametrelerimiz sırasıyla ön bellek tasarım kalıbı, kullanıcı sayısı, rampa süresi ve metottur.

{Gömülü Tasarım Kalıbı, 10, 10, getUserById()}

{Gömülü Dağıtılmış Tasarım Kalıbı, 10, 10, getUserById()}



Şekil 5.4 – Gömülü Tasarım Kalıbı ve Gömülü Dağıtılmış Tasarım Kalıbı için 10 Kullanıcıyla Yapılan Tek Bir Demedi Sorgulayan Sonuçlar

Bu karşılaştırmada 10 iş parçacığı aynı anda uygulama üzerinden istek gönderilerek veri tabanındaki tabloda yer alan bir kaydı getUserById metoduyla çağırmıştır. Bu iş parçacığı kullanıcıları toplam 10 saniyede alınmıştır, ilk saniyede 1, onuncu saniyenin sonunda 10 iş parçacığı kullanıcısı oluşturulmuş ve kayıt çağrılmıştır. Gömülü Tasarım Kalıbı yönteminde veriler ortalama olarak 1,24 ms süresinde gelmektedir. Gömülü Dağıtılmış Tasarım Kalıbı yönteminde ise ortalama olarak 1,53 ms süresinde gelmektedir.

Test 5

Bu kıyaslamada, *getUserById()* metodu için test sonuçları vardır. Test için JMeter programı üzerinde ayarlamalar yapılmıştır. İş parçacığı grubu Apache JMeter üzerinde şu şekilde konfigür edilmiştir:

Thread kullanıcı sayısı: 100

Rampa süresi: 10

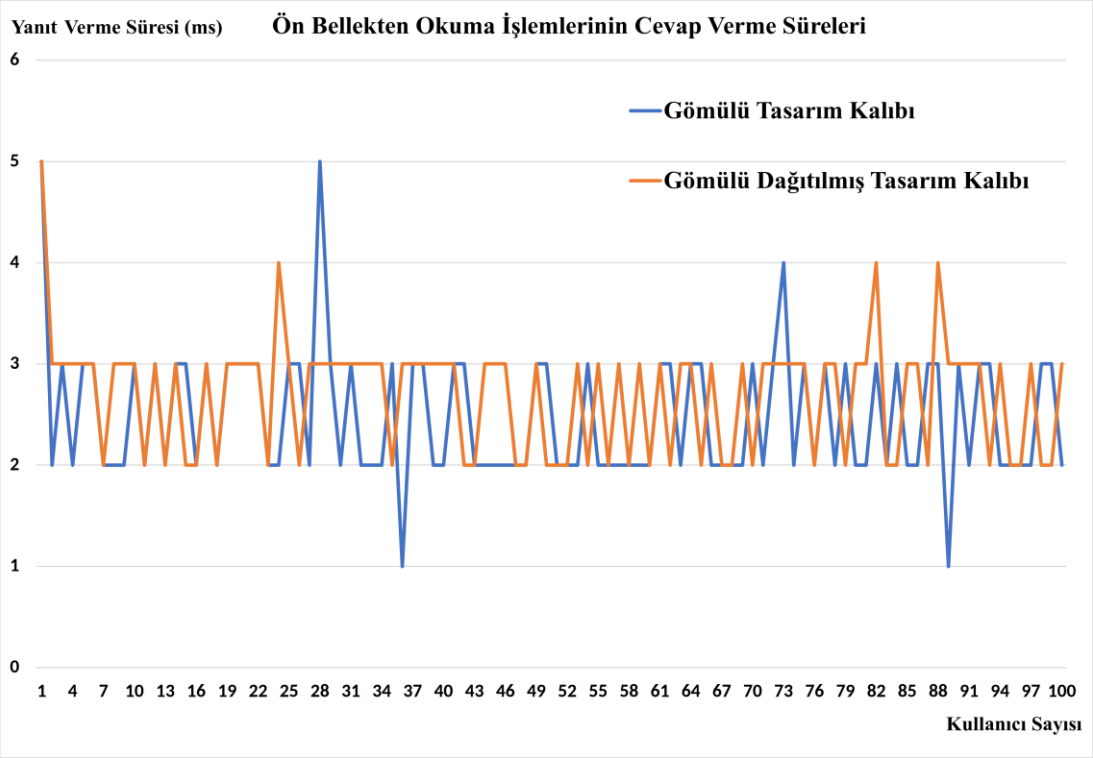
Döngü sayısı: 1

Http İsteği: http://localhost:8080/cache/getUserById/455

Test iki tasarım kalıbının aynı metot için aynı parametrelerle Apache JMeter programındaki konfigürasyonlar seçilerek gerçekleştirilmiştir. Parametrelerimiz sırasıyla ön bellek tasarım kalıbı, kullanıcı sayısı, rampa süresi ve metottur.

{Gömülü Tasarım Kalıbı, 100, 10, getUserById()}

{Gömülü Dağıtılmış Tasarım Kalıbı, 100, 10, getUserById()}



Şekil 5.5 – Gömülü Tasarım Kalıbı ve Gömülü Dağıtılmış Tasarım Kalıbı için 100 Kullanıcıyla Yapılan Tek Bir Demedi Sorgulayan Sonuçlar

Bu karşılaştırmada 100 iş parçacığı aynı anda uygulama üzerinden istek gönderilerek veri tabanındaki tabloda yer alan bir kaydı getUserById metoduyla çağırmıştır. Bu iş parçacığı kullanıcıları toplam 10 saniyede alınmıştır, ilk saniyede 10, onuncu saniyenin sonunda 100 iş parçacığı kullanıcısı oluşturulmuş ve kayıt çağrılmıştır. Gömülü Tasarım Kalıbı yönteminde veriler ortalama olarak 2,48 ms süresinde gelmektedir. Gömülü Dağıtılmış Tasarım Kalıbı yönteminde ise ortalama olarak 2,7 ms süresinde gelmektedir.

Test 6

Bu kıyaslamada, *getUserById()* metodu için test sonuçları vardır. Test için JMeter programı üzerinde ayarlamalar yapılmıştır. İş parçacığı grubu Apache JMeter üzerinde şu şekilde konfigür edilmiştir:

Thread kullanıcı sayısı: 1000

Rampa süresi: 20

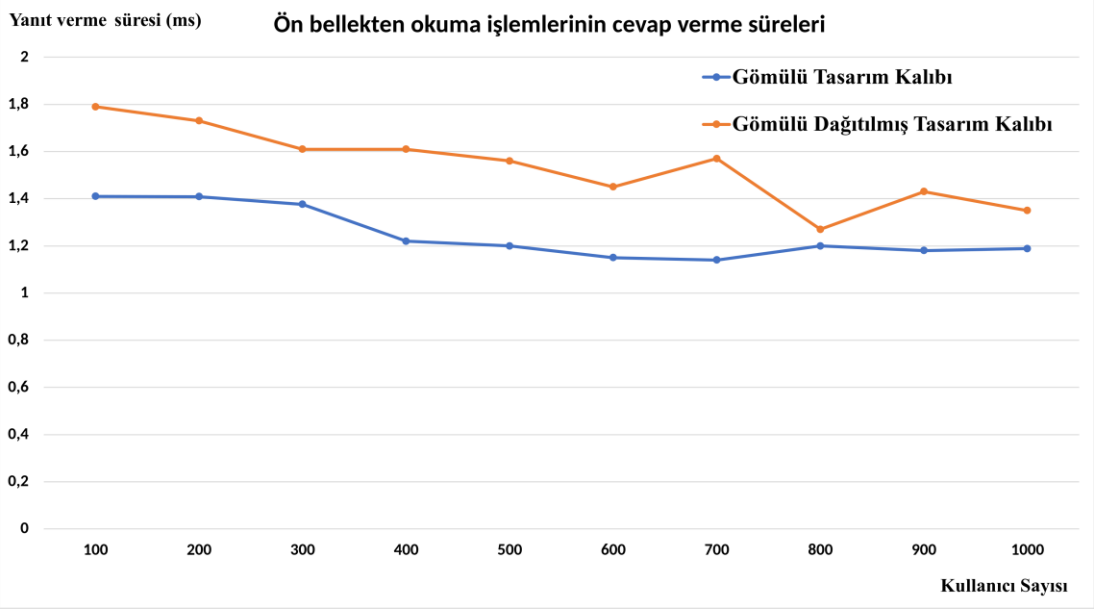
Döngü sayısı: 1

Http İsteği: http://localhost:8080/cache/getUserById/455

Test iki tasarım kalıbının aynı metot için aynı parametrelerle Apache JMeter programındaki konfigürasyonlar seçilerek gerçekleştirilmiştir. Parametrelerimiz sırasıyla ön bellek tasarım kalıbı, kullanıcı sayısı, rampa süresi ve metottur.

{Gömülü Tasarım Kalıbı, 1000, 20, getUserById()}

{Gömülü Dağıtılmış Tasarım Kalıbı, 1000, 20, getUserById()}



Şekil 5.6 – Gömülü Tasarım Kalıbı ve Gömülü Dağıtılmış Tasarım Kalıbı için 1000 Kullanıcıyla Yapılan Tek Bir Demedi Sorgulayan Sonuçlar

Bu karşılaştırmada 1000 iş parçacığı aynı anda uygulama üzerinden istek gönderilerek veritabanındaki tabloda yer alan bir kaydı getUserById metoduyla çağırmıştır. Bu iş parçacığı kullanıcıları toplam 20 saniyede alınmıştır, ilk saniyede 50, onuncu saniyenin sonunda 100 iş parçacığı kullanıcısı oluşturulmuş ve kayıt

çağrılmıştır. Gömülü Tasarım Kalıbı yönteminde veriler ortalama olarak 1,245 ms süresinde gelmektedir. Gömülü Dağıtılmış Tasarım Kalıbı yönteminde ise ortalama olarak 1,537 ms süresinde gelmektedir

Yapılan testler sonucunda *getUserById()* metodu için sonuçlar incelenmiştir. Örnek olarak gösterilen 1000 kayıtlık bir tablodaki verileri ön belleğe alarak okuma işlemi yapan bir CRM uygulamasında tek bir kayıt çağrılmak istenildiğinde Gömülü Tasarım Kalıbı yöntemi Gömülü Dağıtılmış Tasarım Kalıbı yöntemine göre daha hızlı yanıt vermiştir.

Tablo 5.2 – Gömülü Tasarım Kalıbı ve Gömülü Dağıtılmış Tasarım Kalıbı için Tek Bir Demedi Sorgulayan Karşılaştırma

Ön Bellek Tasarım Kalıpları	Kullanıcı Sayısı	Rampa Süresi	Düğüm (Node) Sayısı	Ortalama Yanıt Verme Süresi
Gömülü Tasarım	10	10	1	1,24
Gömülü Dağıtılmış	10	10	2	1,53
Gömülü Tasarım	100	10	1	2,48
Gömülü Dağıtılmış	100	10	2	2,7
Gömülü Tasarım	1000	20	1	1,245
Gömülü Dağıtılmış	1000	20	2	1,537

Karşılaştırma sonucunda Gömülü Dağıtılmış Tasarım Kalıbında instance ve düğüm (node) sayısı fazla olduğundan dolayı maliyeti daha yüksek olmuştur. Gömülü Dağıtılmış Tasarım Kalıbındaki gecikmelerin fazla olmasının nedeni TCP bağlantısında oluşan bağlantı sürelerinin artmasıdır.

Test sonuçları incelendiğinde kullanım senaryolarına göre farklılıklar oluşabilmektedir. Bu farklılıklar, kullanıcıların ne kadar kayıt listelemek istediği ve kaç kullanıcının bu işlemi yapacağıdır. Kıyaslamalarda bu iki parametre göz önüne alınarak tüm kayıtlar çağrılmak istenildiğinde Gömülü Dağıtılmış Tasarım Kalıbının daha verimli ve hızlı sonuç verdiği gözlemlenmiştir. Gömülü Tasarım Kalıbı ise, tek bir kullanıcının bilgilerinin sorgulandığı bir yapıda kullanılabilir. Tek bir kayıt için maliyeti düşük olur ve hızlı sürede getirmektedir. İki farklı mikroservis yapılarak uygulama içerisinde çözüm üretilir. Bu iki yapı dışındaki tasarım kalıplarının sonuçları aşağıdaki test kıyaslamalarında açıklanacaktır.

5.2. İstemci-Sunucu Tasarım Kalıbı ile Bulut (Cloud) Tasarım Kalıbının Kıyaslanması

Testler Jmeter üzerinde yapılmıştır. Bu testler 10, 100, 1000 kullanıcı sayılarına göre oluşturulmuştur. Rampa süresi, kullanıcıların sisteme toplam giriş sürelerini oluşturmaktadır. 10, 100 kullanıcı için 10 saniye rampa süresi yetertli olmuştur. 1000 kullanıcı içinse 20 saniye yeterli olmuştur. Yapılan çalışmalarda iki farklı metot üzerinden kıyaslamalar yapılmıştır. CRM uygulama senaryomuzda 1000 adet kullanıcı bilgilerini içeren kayıt içeren bir tablo tutulmaktadır. Bu tablodan ilk olarak tüm kayıtları getiren ve ön bellekten okuma yapan metot mevcuttur. İkinci metot ise, bu kayıtlardan sadece bir kaydı getiren metottur. Karşılaştırmalar bu metotlar üzerinden yapılmıştır. İstemci-Sunucu Tasarım Kalıbı ve Bulut Tasarım Kalıbı testlerinde JMeter üzerinde HttpRequest bileşeni kullanılarak Hazelcast ön bellek yapısına istek göndererek cevap verme süreleri ölçülmüştür. Sonuçlar excel dosyasına yazılmıştır. Bağlantı süreleri ve gecikme süreleri ele alınarak cevap verme süreleri üzerinden yorumlanmıştır.

Test 7

Bu kıyaslamada, *getAllUsers()* metodu için test sonuçları vardır. Test için JMeter programı üzerinde ayarlamalar yapılmıştır. İş parçacığı grubu Apache JMeter üzerinde şu şekilde konfigür edilmiştir:

Thread kullanıcı sayısı: 10

Rampa süresi: 10

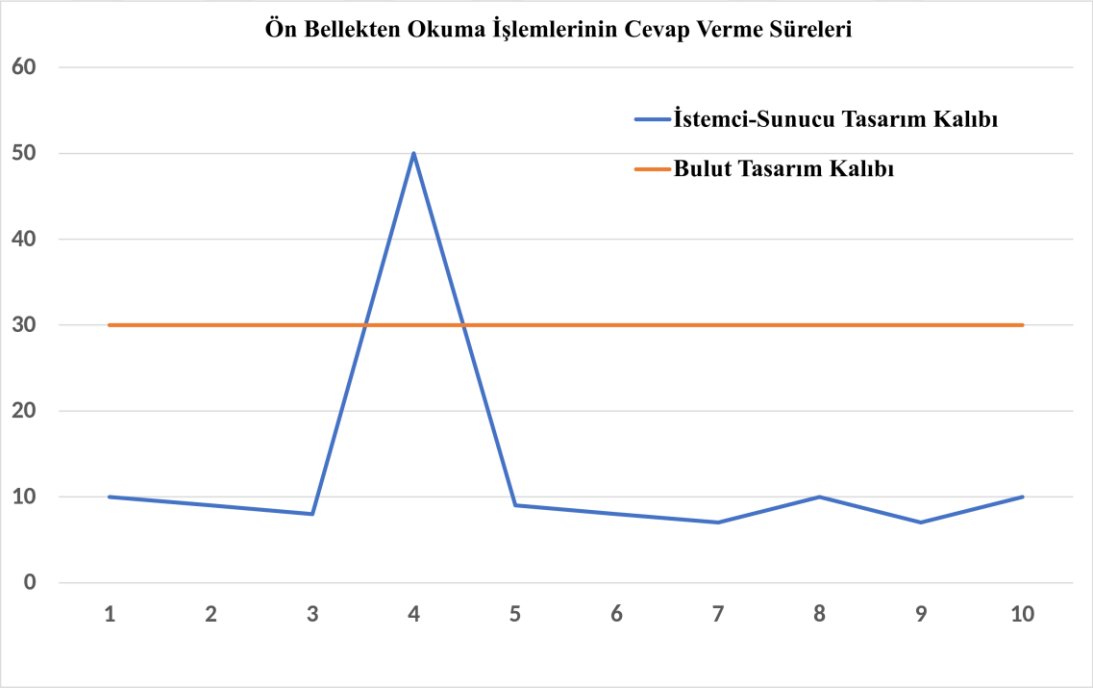
Döngü sayısı: 1

Http İsteği: http://localhost:8080/cache/getUserById/455

Test iki tasarım kalıbının aynı metot için aynı parametrelerle Apache JMeter programındaki konfigürasyonlar seçilerek gerçekleştirilmiştir. Parametrelerimiz sırasıyla ön bellek tasarım kalıbı, kullanıcı sayısı, rampa süresi ve metottur.

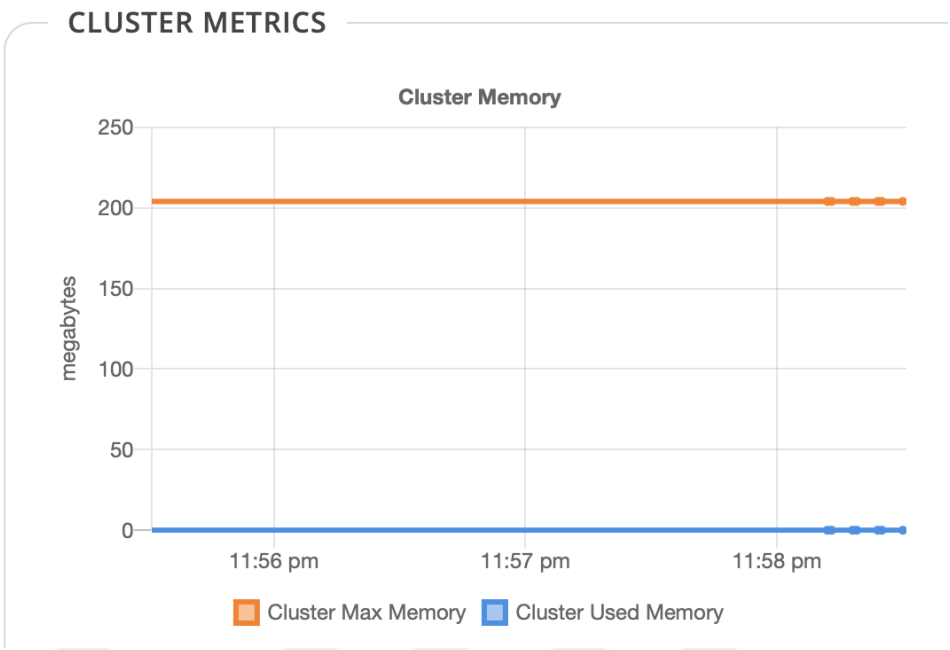
{İstemci-Sunucu Tasarım Kalıbı, 10, 10, getAllUsers()}

{Bulut Tasarım Kalıbı, 10, 10, getAllUsers()}



Şekil 5.7 – İstemci-Sunucu Tasarım Kalıbı ve Bulut Tasarım Kalıbı için 10 Kullanıcıyla Yapılan Tüm Tabloyu Sorgulayan Sonuçlar

Şekil 5.7'deki karşılaştırmada 10 iş parçacığı aynı anda uygulama üzerinden istek gönderilerek veri tabanındaki tabloda yer alan 1000 kaydı çağırmıştır. İstemci-Sunucu Tasarım Kalıbı yönteminde veriler ortalama olarak 12,8ms süresinde gelmektedir. Bulut Tasarım Kalıbı yönteminde ise veriler ortalama olarak 30ms süresinde gelmektedir. İstemci-Sunucu Tasarım Kalıbı yönteminde tek bir member kullanılır. İstemci-Sunucu Tasarım Kalıbı yönteminde, Hazelcast yapısı sunucu (server) olarak bulunmaktadır. Uygulama isteklerini server'a iletir ve server'ın veritabanından aldığı sonuçları ön bellek üzerinden dönmesini bekler. Burada yapılabilecek parametre ayarları server'ın fiziksel özelliklerini arttırmak veya azaltmak olabilir. Bulut Tasarım Kalıbı yapısında ise, uygulama dışarıdaki yani ayrı bir fiziksel makinedeki sunucuya gider. Bulut Tasarım Kalıbı ortamına bağlanma süresinden kaynaklı bir gecikme olmuştur. İstemci-Sunucu Tasarım Kalıbı ve diğer yöntemlere göre oluşan gecikme bağlantı süresinden kaynaklanmaktadır. Bu modelde Hazelcast Cloud teknolojisi kullanılmıştır. Platform üzerinde test-cluster adında bir cluster yapısı oluşturulmuştur. Uygulamada konfigürasyon buradaki bilgilere göre yapılmıştır. Güvenlik açısından İstemci-Sunucu Tasarım Kalıbı'na göre farkı bu cluster'ın üretmiş olduğu benzersiz bir key (anahtar) değeridir. Bu değer uygulamada yazılmıştır ve erişim için gerekli bir değer durumuna gelmiştir. Bulut Tasarım Kalıbı yönteminde performans incelemeleri yapılabilir. Uygulama çalışırken 10 kullanıcı için yapılan testte sistemin memory, CPU ölçümleri platform üzerinden izlenebilir.



Şekil 5.8 – Bulut Tasarım Kalıbındaki Küme Yapısının Ram Bilgileri

Bu çalışmada oluşturulan küme 200 MB RAM ayırmıştır. İşlemin gerçekleştiği saatlerde maksimum RAM'e kadar kullanılabilir hale gelmiştir. Kıyaslamalarda 10 kullanıcı için yapılan tüm kayıtları getirme işleminde İstemci-Sunucu Tasarım Kalıbının Bulut Tasarım Kalıbı'na göre ortalama 19,8ms hızlı sonuç getirdiği görülmüştür. Bu durumun kaynağı Java Sanal Makinesi (Java Virtual Machine) yapısının İstemci-Sunucu Tasarım Kalıbında aynı makinede olmasıdır.

Test 8

Bu kıyaslamada, *getAllUsers()* metodu için test sonuçları vardır. Test için JMeter programı üzerinde ayarlamalar yapılmıştır. İş parçacığı grubu Apache JMeter üzerinde şu şekilde konfigür edilmiştir:

Thread kullanıcı sayısı: 100

Rampa süresi: 10

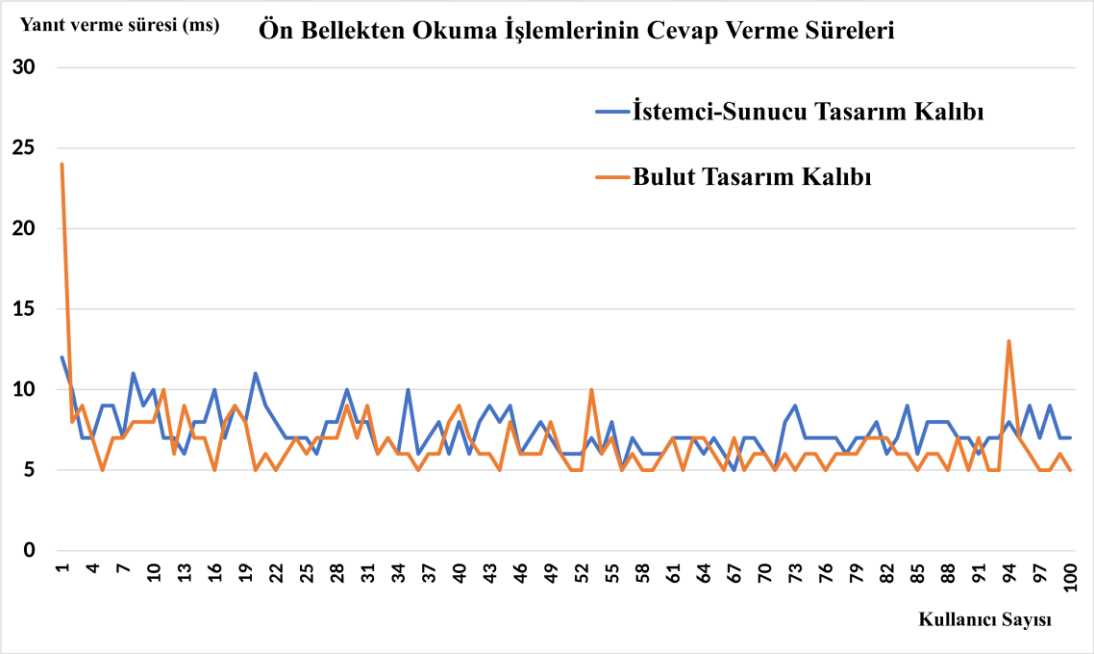
Döngü sayısı: 1

Http İsteği: http://localhost:8080/cache/getUserById/455

Test iki tasarım kalıbının aynı metot için aynı parametrelerle Apache JMeter programındaki konfigürasyonlar seçilerek gerçekleştirilmiştir. Parametrelerimiz sırasıyla ön bellek tasarım kalıbı, kullanıcı sayısı, rampa süresi ve metottur.

{İstemci-Sunucu Tasarım Kalıbı, 100, 10, getAllUsers()}

{Bulut Tasarım Kalıbı, 100, 10, getAllUsers()}

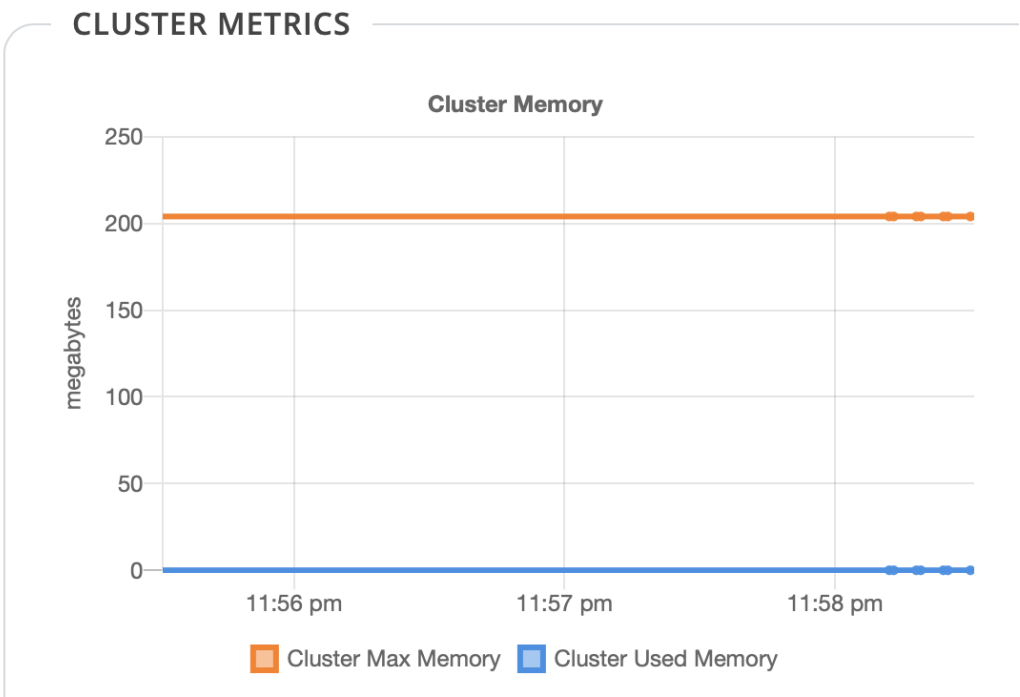


Şekil 5.9 – İstemci-Sunucu Tasarım Kalıbı ve Bulut Tasarım Kalıbı için 100 Kullanıcıyla Yapılan Tüm Tabloyu Sorgulayan Sonuçlar

Bu karşılaştırmada 100 iş parçacığı aynı anda uygulama üzerinden istek gönderilerek veri tabanındaki tabloda yer alan 1000 kaydı çağırmıştır. İstemci-Sunucu Tasarım Kalıbı yönteminde veriler ortalama olarak 7,42ms süresinde gelmektedir.

Bulut Tasarım Kalıbı yönteminde ise veriler ortalama olarak 6,67ms süresinde gelmektedir. İstemci-Sunucu Tasarım Kalıbı yönteminde tek bir member kullanılır, Bulut Tasarım Kalıbı yönteminde ise iki tane farklı member cluster yapısı oluşturur ve Hazelcast'ın multicast özelliğini kullanarak verileri ön bellekten daha hızlı sürede getirmiş olur. İstemci-Sunucu Tasarım Kalıbı yönteminde, Hazelcast yapısı server olarak bulunmaktadır. Uygulama isteklerini server'a iletir ve server'ın veritabanından aldığı sonuçları ön bellek üzerinden dönmesini bekler. Burada yapılabilecek parametre ayarları server'ın fiziksel özelliklerini arttırmak veya azaltmak olabilir. Bulut Tasarım Kalıbı yapısında ise, uygulama dışarıdaki yani ayrı bir fiziksel makinedeki sunucuya gider.

Yapılan kıyaslamalarda, Bulut Tasarım Kalıbı yönteminde ortalama 0,75ms daha hızlı yanıt vermiştir. Bu test sonucu bir önceki testteki gibi ilk 10 kullanıcı için Bulut Tasarım Kalıbının daha uzun sürede yanıt verdiği görülmüştür. Bulut Tasarım Kalıbı yönteminde platformun yönetimini takip etmek gerekir, ölçeklendirme ve yedekleme duruma göre yapılabilir. Bu durumda oluşan memory grafiği aşağıda verilmiştir.



Şekil 5.10 – Bulut Tasarım Kalıbının Küme Yapısının Ram Bilgileri

Test 9

Bu kıyaslamada, *getAllUsers()* metodu için test sonuçları vardır. Test için JMeter programı üzerinde ayarlamalar yapılmıştır. İş parçacığı grubu Apache JMeter üzerinde şu şekilde konfigür edilmiştir:

Thread kullanıcı sayısı: 1000

Rampa süresi: 20

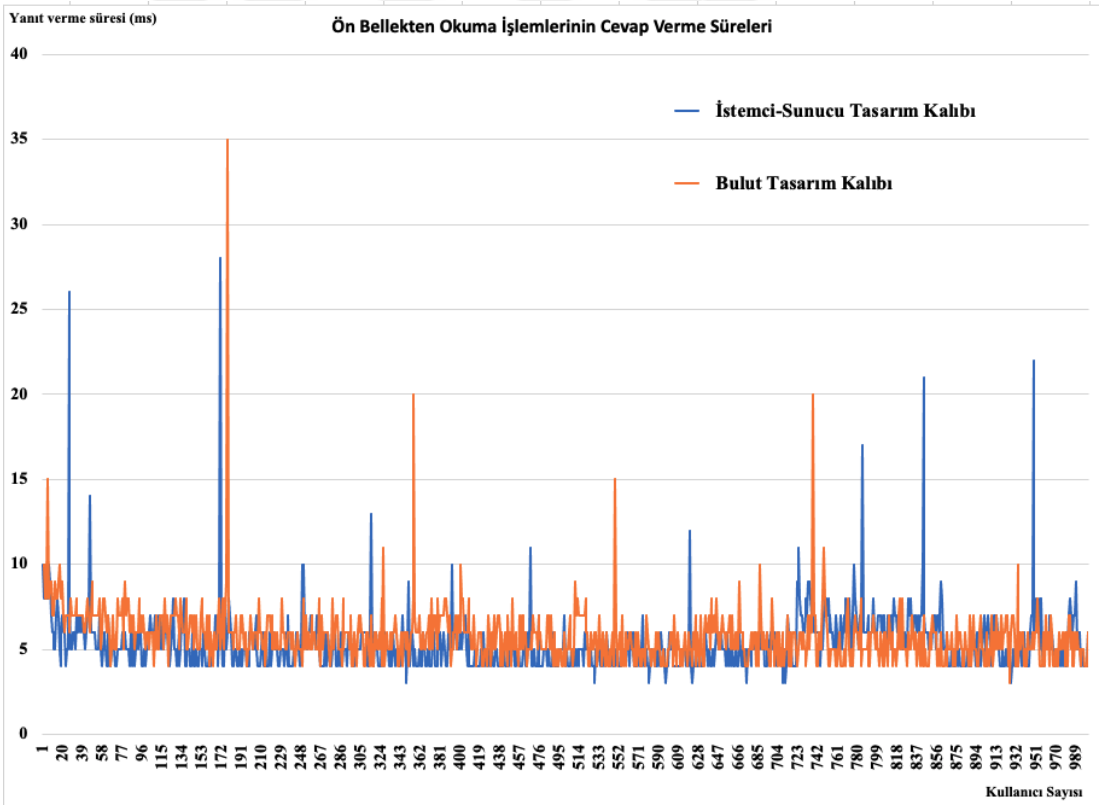
Döngü sayısı: 1

Http İsteği: http://localhost:8080/cache/getUserById/455

Test iki tasarım kalıbının aynı metot için aynı parametrelerle Apache JMeter programındaki konfigürasyonlar seçilerek gerçekleştirilmiştir. Parametrelerimiz sırasıyla ön bellek tasarım kalıbı, kullanıcı sayısı, rampa süresi ve metottur.

{İstemci-Sunucu Tasarım Kalıbı, 1000, 20, getAllUsers()}

{Bulut Tasarım Kalıbı, 1000, 20, getAllUsers()}



Şekil 5.11 – İstemci-Sunucu Tasarım Kalıbı ve Bulut Tasarım Kalıbı için 1000 Kullanıcıyla Yapılan Tüm Tabloyu Sorgulayan Sonuçlar

Bu karşılaştırmada 1000 iş parçacığı aynı anda uygulama üzerinden istek gönderilerek veri tabanındaki tabloda yer alan 1000 kaydı çağırmıştır. İstemci-Sunucu Tasarım Kalıbı yönteminde veriler ortalama olarak 5,34ms süresinde gelmektedir.

Bulut Tasarım Kalıbı yönteminde ise veriler ortalama olarak 5,75ms süresinde gelmektedir. İstemci-Sunucu Tasarım Kalıbı yönteminde tek bir member kullanılır. İstemci-Sunucu Tasarım Kalıbı yönteminde, Hazelcast yapısı server olarak bulunmaktadır. Uygulama isteklerini server'a iletir ve server'ın veritabanından aldığı sonuçları ön bellek üzerinden dönmesini bekler. Burada yapılabilecek parametre ayarları server'ın fiziksel özelliklerini arttırmak veya azaltmak olabilir. Bulut Tasarım Kalıbı yapısında ise, uygulama dışarıdaki yani ayrı bir fiziksel makinedeki sunucuya gider.

Yapılan testler sonucunda *getAllUsers()* metodu için sonuçlar incelenmiştir. Örnek olarak gösterilen 1000 kayıtlık bir tablodaki verileri ön belleğe alarak okuma işlemi yapan bir CRM uygulamasında tek bir kayıt çağrılmak istenildiğinde İstemci-Sunucu Tasarım Kalıbı ve Bulut Tasarım Kalıbı yöntemlerinin yanıt verme süreleri aşağıda belirtilmiştir.

Tablo 5.3 – İstemci-Sunucu Tasarım Kalıbı ve Bulut Tasarım Kalıbı için Tüm Tabloyu Sorgulayan Karşılaştırma

Ön Bellek Tasarım Kalıpları	Kullanıcı Sayısı	Rampa Süresi	Node Sayısı	Ortalama Yanıt Verme Süresi
İstemci-Sunucu	10	10	1	12,8
Bulut Tasarım	10	10	2	30
İstemci-Sunucu	100	10	1	7,42
Bulut Tasarım	100	10	2	6,67
İstemci-Sunucu	1000	20	1	5,34
Bulut Tasarım	1000	20	2	5,75

Bulut Tasarım Kalıbı yönteminde, 10 kullanıcılı testte 12ms gecikme yaşanmıştır. Bunun sonucunda yapılan ilk testteki ortalama yanıt verme süresi yüksek çıkmıştır. Bir sonraki testte 100 kullanıcı için de ilk kullanıcı girerken 17ms gecikme yaşanmıştır, sonrasındaki gecikme süreleri İstemci-Sunucu Tasarım Kalıbına göre azaldığı için ortalama yanıt verme süresi azalmıştır. 1000 kullanıcılı yapılan testte ise iki farklı nokta gözlemlenmiştir. Bu noktalarda 178. Ve 356. kullanıcılar sisteme girerken 3ms ve 4ms gecikme yaşanmıştır. Bu değer 1000 kullanıcılı sistem içerisinde ortalamaya göre yüksek bir süredir. İstemci-Sunucu Tasarım Kalıbı yönteminde bu gibi gecikmeler sadece ilk kullanıcının sisteme giriş aşamasında yaşanmıştır.

Test 10

Bu kıyaslamada, *getUserById()* metodu için test sonuçları vardır. Test için JMeter programı üzerinde ayarlamalar yapılmıştır. İş parçacığı grubu Apache JMeter üzerinde şu şekilde konfigür edilmiştir:

Thread kullanıcı sayısı: 1000

Rampa süresi: 10

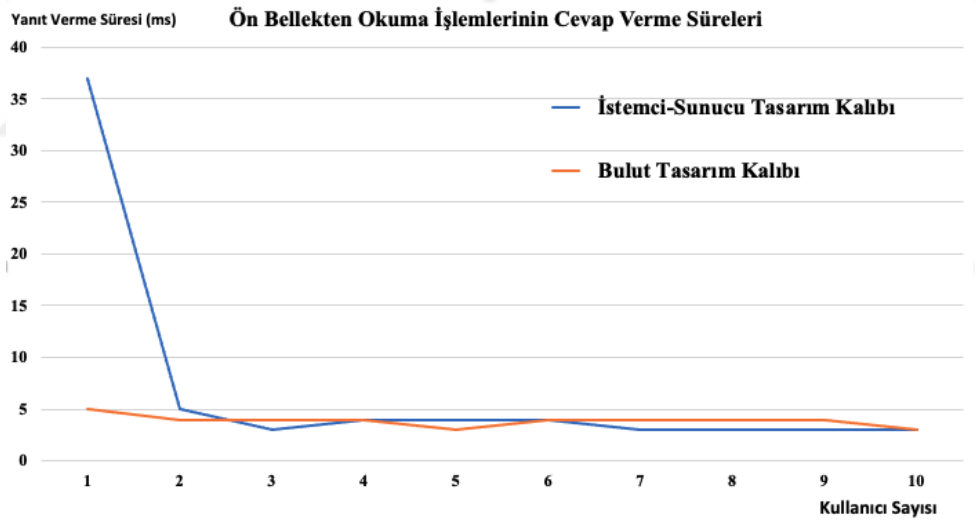
Döngü sayısı: 1

Http İsteği: http://localhost:8080/cache/getUserById/455

Test iki tasarım kalıbının aynı metot için aynı parametrelerle Apache JMeter programındaki konfigürasyonlar seçilerek gerçekleştirilmiştir. Parametrelerimiz sırasıyla ön bellek tasarım kalıbı, kullanıcı sayısı, rampa süresi ve metottur.

{İstemci-Sunucu Tasarım Kalıbı, 10, 10, getUserById()}

{Bulut Tasarım Kalıbı, 10, 10, getUserById()}



Şekil 5.12 – İstemci-Sunucu Tasarım Kalıbı ve Bulut Tasarım Kalıbı için 10 Kullanıcıyla Yapılan Tek Bir Demedi Sorgulayan Sonuçlar

Bu karşılaştırmada 10 iş parçacığı aynı anda uygulama üzerinden istek gönderilerek veri tabanındaki tabloda yer alan tek bir kaydı *getUserById()* metoduyla çağırmıştır. İstemci-Sunucu Tasarım Kalıbı yönteminde veriler ortalama olarak 6,9ms süresinde gelmektedir. Bulut Tasarım Kalıbı yönteminde ise veriler ortalama olarak 3,9ms süresinde gelmektedir. İstemci-Sunucu Tasarım Kalıbı yönteminde tek bir member kullanılır. İstemci-Sunucu Tasarım Kalıbı yönteminde, Hazelcast yapısı server olarak

bulunmaktadır. Uygulama isteklerini server'a iletir ve server'ın veritabanından aldığı sonuçları ön bellek üzerinden dönmesini bekler. Burada yapılabilecek parametre ayarları server'ın fiziksel özelliklerini arttırmak veya azaltmak olabilir. İstemci-Sunucu Tasarım Kalıbı ve diğer yöntemlere göre oluşan gecikme bağlantı süresinden kaynaklanmaktadır. Yapılan incelemelerde tek bir kayıt için gecikme sürelerinin Bulut Tasarım Kalıbında daha az olduğu gözlemlenmiştir. İstemci-Sunucu Tasarım Kalıbı yönteminde daha önceki testlerde olduğu gibi sunucuya bağlanırken ilk kullanıcının sisteme giriş süresinde ortalamanın çok üzerinde bir gecikme süresi olmuştur. Çok kullanıcılı sistemlerde bu ortalama süre ilerleyen kullanıcılarda azalmaktadır fakat az kullanıcılı bir sistemde ortalama süreye etkisi çok fazladır.

Test 11

Bu kıyaslamada, *getUserById()* metodu için test sonuçları vardır. Test için JMeter programı üzerinde ayarlamalar yapılmıştır. İş parçacığı grubu Apache JMeter üzerinde şu şekilde konfigür edilmiştir:

Thread kullanıcı sayısı: 1000

Rampa süresi: 10

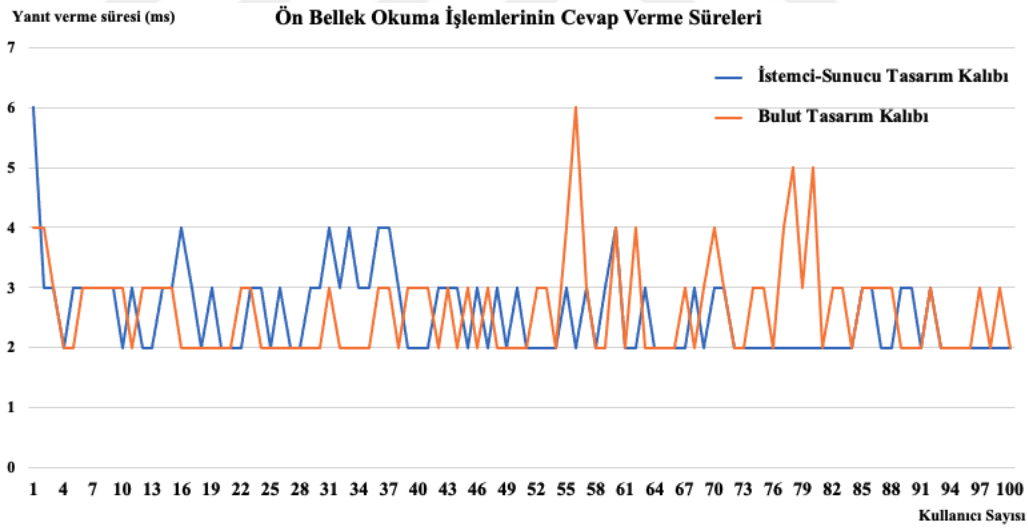
Döngü sayısı: 1

Http İsteği: http://localhost:8080/cache/getUserById/455

Test iki tasarım kalıbının aynı metot için aynı parametrelerle Apache JMeter programındaki konfigürasyonlar seçilerek gerçekleştirilmiştir. Parametrelerimiz sırasıyla ön bellek tasarım kalıbı, kullanıcı sayısı, rampa süresi ve metottur.

{İstemci-Sunucu Tasarım Kalıbı, 100, 10, getUserById()}

{Bulut Tasarım Kalıbı, 100, 10, getUserById()}



Şekil 5.13 – İstemci-Sunucu Tasarım Kalıbı ve Bulut Tasarım Kalıbı için 100 Kullanıcıyla Yapılan Tek Bir Demedi Sorgulayan Sonuçlar

Bu karşılaştırmada 100 iş parçacığı aynı anda uygulama üzerinden istek gönderilerek veri tabanındaki tabloda yer alan tek bir kaydı *getUserById()* metoduyla çağırmıştır. İstemci-Sunucu Tasarım Kalıbı yönteminde veriler ortalama olarak 2,55ms süresinde

gelmektedir. Bulut Tasarım Kalıbı yönteminde ise veriler ortalama olarak 2,63ms süresinde gelmektedir. İstemci-Sunucu Tasarım Kalıbı yönteminde tek bir member kullanılır. İstemci-Sunucu Tasarım Kalıbı yönteminde, Hazelcast yapısı server olarak bulunmaktadır. Uygulama isteklerini server'a iletir ve server'ın veritabanından aldığı sonuçları ön bellek üzerinden dönmesini bekler. Burada yapılabilecek parametre ayarları server'ın fiziksel özelliklerini arttırmak veya azaltmak olabilir. İstemci-Sunucu Tasarım Kalıbı ve diğer yöntemlere göre oluşan gecikme bağlantı süresinden kaynaklanmaktadır. Yapılan incelemelerde, ortalama gecikme süresinin Bulut Tasarım Kalıbında daha fazla olduğu gözlemlenmiştir. Buna ek olarak bağlantı süresi değeri diğer testlerdeki gibi ortalama 70-80 kullanıcı sisteme girdiğinde artmıştır. Bağlantı süresinin gecikmeye etkisinden kaynaklı 57. kullanıcının sisteme girişi 6ms olmuştur. Bu durum Bulut Tasarım Kalıbının daha geç yanıt vermesine sebep olmuştur.

Test 12

Bu kıyaslamada, *getUserById()* metodu için test sonuçları vardır. Test için JMeter programı üzerinde ayarlamalar yapılmıştır. İş parçacığı grubu Apache JMeter üzerinde şu şekilde konfigür edilmiştir:

Thread kullanıcı sayısı: 1000

Rampa süresi: 10

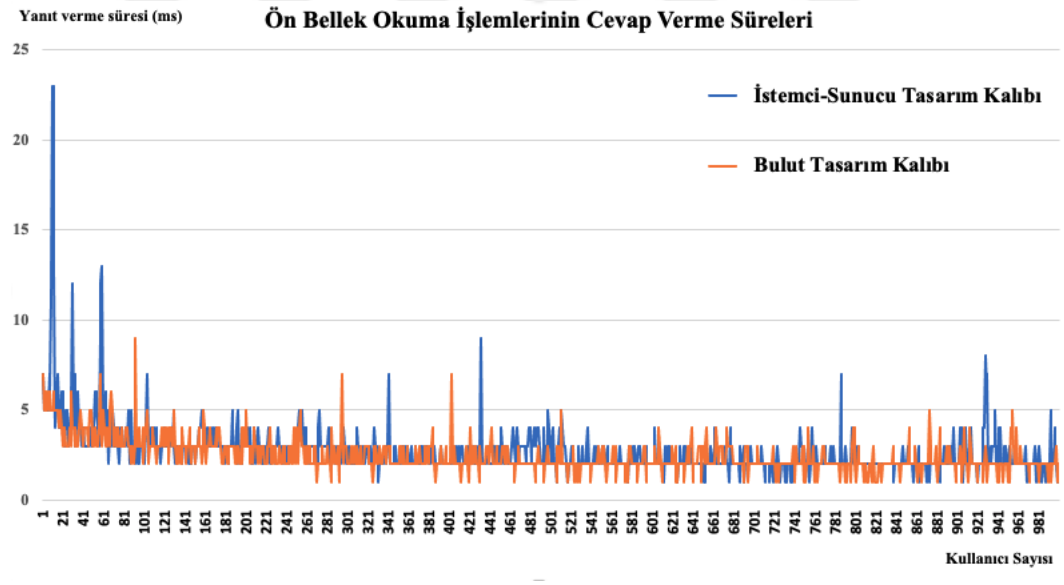
Döngü sayısı: 1

Http İsteği: http://localhost:8080/cache/getUserById/455

Test iki tasarım kalıbının aynı metot için aynı parametrelerle Apache JMeter programındaki konfigürasyonlar seçilerek gerçekleştirilmiştir. Parametrelerimiz sırasıyla ön bellek tasarım kalıbı, kullanıcı sayısı, rampa süresi ve metottur.

{İstemci-Sunucu Tasarım Kalıbı, 1000, 20, getUserById()}

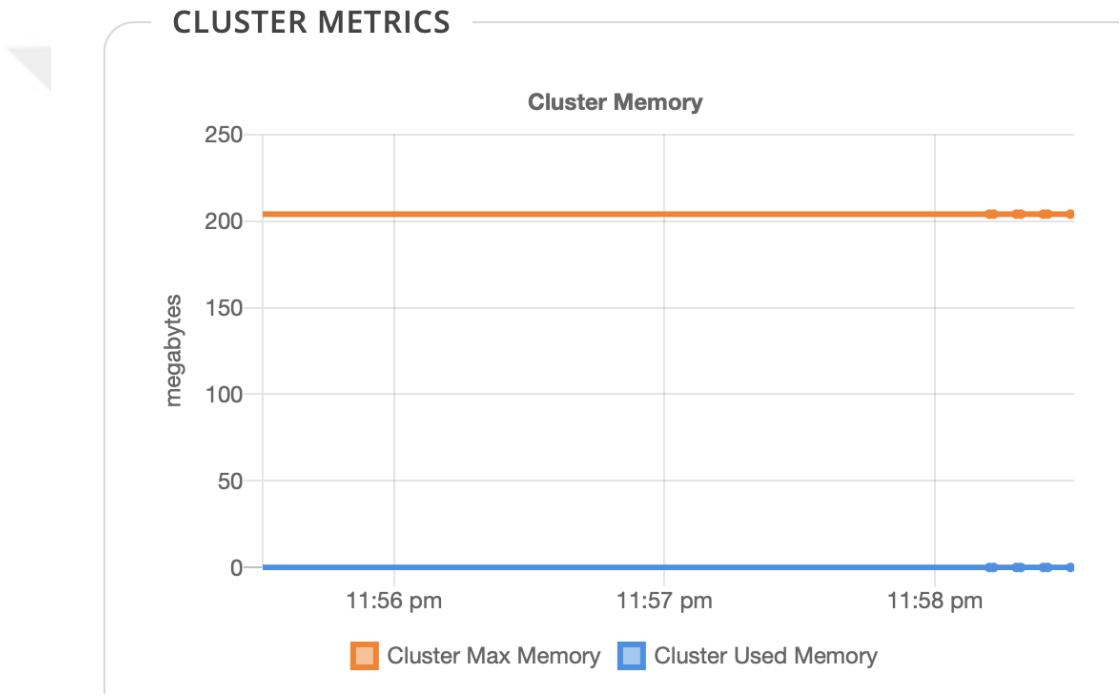
{Bulut Tasarım Kalıbı, 1000, 20, getUserById()}



Şekil 5.14 – İstemci-Sunucu Tasarım Kalıbı ve Bulut Tasarım Kalıbı için 1000 Kullanıcıyla Yapılan Tek Bir Demedi Sorgulayan Sonuçları

Bu karşılaştırmada 1000 iş parçacığı aynı anda uygulama üzerinden istek gönderilerek veri tabanındaki tabloda yer alan tek bir kaydı *getUserById()* metoduyla çağırmıştır. İstemci-Sunucu Tasarım Kalıbı yönteminde veriler ortalama olarak 2,72ms süresinde gelmektedir. Bulut Tasarım Kalıbı yönteminde ise veriler ortalama olarak 2,45ms

süresinde gelmektedir. İstemci-Sunucu Tasarım Kalıbı yönteminde tek bir member kullanılır. İstemci-Sunucu Tasarım Kalıbı yönteminde, Hazelcast yapısı server olarak bulunmaktadır. Uygulama isteklerini server'a iletir ve server'ın veritabanından aldığı sonuçları ön bellek üzerinden dönmesini bekler. Testin sonucu ilk 100 kullanıcı için bir önceki test ile doğru orantı göstermiştir. Grafikte de görüldüğü üzere İstemci-Sunucu Tasarım Kalıbı yönteminde ilk 100 kullanıcının sisteme giriş süresi artmıştır. Bulut Tasarım Kalıbı modeli ortalama olarak 0,27ms daha hızlı sonuç getirmiş oldu. Sistem metrikleri incelendiğinde cluster yapısı 200MB hafıza ayırmıştır.



Şekil 5.14 – Bulut Tasarım Kalıbı Küme Yapısının Ram Bilgileri

İstemci-Sunucu Tasarım Kalıbı yöntemi geçmişten günümüze kullanılan klasik bir yöntemdir. Bu yöntemin son yıllarda benzer bir yapısı Bulut Tasarım Kalıbı modelidir. Her iki model de tasarlanırken aynı programlama dili ile kodlanmıştır. Aynı veritabanındaki kayıtları sorgulamıştır. İstemci-Sunucu Tasarım Kalıbı ile ilgili yapılan tüm testlerde ilk kullanıcıların sisteme giriş süreleri daha önceki Gömülü Tasarım Kalıbı ve Gömülü Dağıtılmış Tasarım Kalıbı testlerine göre, Bulut Tasarım Kalıbı testlerine göre ortalama olarak daha fazla çıkmıştır.

Yapılan testlerin tümü incelendiğinde, aşağıdaki tabloda görüldüğü üzere kullanıcı sayılarının çok olduğu senaryoda Bulut Tasarım Kalıbının süre olarak İstemci-Sunucu Tasarım Kalıbına göre yakın olduğu ortaya çıkmıştır.

Tablo 5.4 – İstemci-Sunucu Tasarım Kalıbı ve Bulut Tasarım Kalıbı için Tek Bir Demedi Sorgulayan Karşılaştırma

Ön Bellek Tasarım Kalıpları	Kullanıcı Sayısı	Rampa Süresi	Düğüm (Node) Sayısı	Ortalama Yanıt Verme Süresi
İstemci-Sunucu	10	10	1	6,9
Bulut Tasarım	10	10	2	3,9
İstemci-Sunucu	100	10	1	2,55
Bulut Tasarım	100	10	2	2,63
İstemci-Sunucu	1000	20	1	2,72
Bulut Tasarım	1000	20	2	2,45

5.3. Sepet Tasarım Kalıbının Gömülü Dağıtılmış ve İstemci-Sunucu Tasarım Kalıplarıyla Kıyaslanması

Testler Jmeter üzerinde yapılmıştır. Bu testler 10 ve 1000 kullanıcı sayılarına göre oluşturulmuştur. Rampa süresi, kullanıcıların sisteme toplam giriş sürelerini oluşturmaktadır. 10 kullanıcı için 10 saniye rampa süresi yetertli olmuştur. 1000 kullanıcı içinse 20 saniye yeterli olmuştur. Yapılan çalışmalarda iki farklı metot üzerinden kıyaslamalar yapılmıştır. CRM uygulama senaryomuzda 1000 adet kullanıcı bilgilerini içeren kayıt içeren bir tablo tutulmaktadır. Bu tablodan ilk olarak tüm kayıtları getiren ve ön bellekten okuma yapan metot mevcuttur. İkinci metot ise, bu kayıtlardan sadece bir kaydı getiren metottur. Karşılaştırmalar bu metotlar üzerinden yapılmıştır. Gömülü Dağıtılmış Tasarım Kalıbı ve Bulut Tasarım Kalıbı testlerinde JMeter üzerinde HttpRequest bileşeni kullanılarak Hazelcast ön bellek yapısına istek göndererek cevap verme süreleri ölçülmüştür. Sepet Tasarım Kalıbında ise thread (iş parçacığı) yapısı kullanılarak test edilmiştir. Sonuçlar excel dosyasına yazılmıştır. Sepet Tasarım Kalıbı yönteminde iki farklı mikroservis ve iki farklı Hazelcast modeli (bağlamı) kullanılmıştır. Mikroservislerden birisi Gömülü Dağıtılmış Tasarım Kalıbı mimari yapısındadır, diğeri ise Bulut Tasarım Kalıbı mimari yapısındadır. Modellerden birincisi hazelcast-sidecar-0, ikincisiyse hazelcast-sidecar-1'dir. Yapılan karşılaştırmalarda üç pattern ele alınmıştır ve metotların yanıt verme süresi üzerinden karşılaştırma yapılmıştır.

5.3.1. Tüm Tabloyu Sorgulayan Test Sonuçlarının Karşılaştırılması

Yapılan çalışmada Sepet Tasarım Kalıbının daha önceki test sonuçlarında ortaya çıkmış olan Gömülü Dağıtılmış Tasarım Kalıbı ve Bulut Tasarım Kalıbıyla karşılaştırılması yapılmıştır. Daha önceden testleri yapılmış olan bu iki modelin birisi dağıtılmış yapıda bir ön bellek, diğeri ise bulut platformunda var olan bir ön bellek yapısı kullanmıştır. Sepet Tasarım Kalıbı (Sidecar Pattern) modelde iki mikroservis uygulaması bu yapıları kullanmıştır. Test sonuçlarında Sepet Tasarım Kalıbı yapısı için kullanıcı sayılarının 10 olduğu durumda iki yönteme göre tüm kayıtları ortalama olarak 13ms daha hızlı getirmiştir. Diğer kıyaslama olan 1000 kullanıcılı senaryoda ise, Bulut Tasarım Kalıbına göre 1,74ms daha geç, Gömülü Dağıtılmış Tasarım Kalıbına göre ise 30,8ms daha hızlı getirmiştir. Bu farklılığın nedeni Sepet Tasarım Kalıbı yapısında 1000 kullanıcının sisteme giriş süresi ve oluşan gecikme süresinin Bulut Tasarım Kalıbına göre daha fazla olmasıdır. İsteğin Kubernetes pod'larında yer alan uygulamalara iletilmesi yavaşlığa sebep olmuştur.

Test üç farklı tasarım kalıbının aynı metot için aynı parametrelerle Apache JMeter programındaki konfigürasyonlar seçilerek gerçekleştirilmiştir. Parametrelerimiz sırasıyla ön bellek tasarım kalıbı, kullanıcı sayısı, rampa süresi ve metottur.
{Gömülü Dağıtılmış, (10,100,1000), (10,10,20), getAllUsers()}
{Bulut Tasarım Kalıbı, (10,100,1000), (10,10,20), getAllUsers()}
{Sepet Tasarım Kalıbı, (10,100,1000), (10,10,20), getAllUsers()}

Tablo 5.5 – Sepet Tasarım Kalıbı ve Diğer Tasarım Kalıpları için Tüm Tabloyu Sorgulayan Karşılaştırma

Ön Bellek Tasarım Kalıpları	Kullanıcı Sayısı	Rampa Süresi	Düğüm (Node) Sayısı	Ortalama Yanıt Verme Süresi
Gömülü Dağıtılmış	10	10	1	33,6
Bulut Tasarım	10	10	2	30
Sepet Tasarım	10	10	2	17,44
Gömülü Dağıtılmış	1000	20	1	37,57
Bulut Tasarım	1000	20	2	5,75
Sepet Tasarım	1000	20	2	7,49

5.3.2. Tek Bir Demedi (Tuple) Sorgulayan Test Sonuçlarının Karşılaştırılması

Yapılan çalışmada Sepet Tasarım Kalıbının daha önceki test sonuçlarında ortaya çıkmış olan Gömülü Dağıtılmış Tasarım Kalıbı ve Bulut Tasarım Kalıbı karşılaştırılması yapılmıştır. Daha önceden testleri yapılmış olan bu iki modelin birisi dağıtılmış yapıda bir ön bellek, diğeri ise bulut platformunda var olan bir ön bellek yapısı kullanmıştır. Sepet Tasarım Kalıbı modelinde iki mikroservis uygulaması bu yapıları kullanmıştır. Tek bir kayıt getirme işleminde 10 kullanıcı için en ideal yöntem Sepet Tasarım Kalıbı (Sidecar Pattern) olmuştur. 1000 kullanıcıyla yapılan testte ise kullanıcıların istek anında sisteme bağlantı süresinden kaynaklı bir gecikme yaşanmıştır. Bu durum rampa süresi arttırılarak çözülebilir fakat bu durumda da bekleme süresi artmaktadır.

Tablo 5.6 – Sepet Tasarım Kalıbı ve Diğer Tasarım Kalıpları için Tek Bir Demedi Sorgulayan Karşılaştırma

Ön Bellek Tasarım Kalıpları	Kullanıcı Sayısı	Rampa Süresi	Düğüm (Node) Sayısı	Ortalama Yanıt Verme Süresi
Gömülü Dağıtılmış	10	10	1	1,53
Bulut Tasarım	10	10	2	3,9
Sepet Tasarım	10	10	2	1,3
Gömülü Dağıtılmış	1000	20	1	1,537
Bulut Tasarım	1000	20	2	2,45
Sepet Tasarım	1000	20	2	1,9

6. TARTIŞMA

Yapılan çalışmada mikroservis ekosisteminde bugüne kadar yapılan ön bellekle ilgili ve servislerin durum yönetimleriyle ilgili çalışmalar incelenmiştir. Mikroservis mimarileri içerisinde geçmişten günümüze sıklıkla kullanılan Gömülü Tasarım Kalıbı, Gömülü Dağıtılmış Tasarım Kalıbı, İstemci-Sunucu Tasarım Kalıbı, Bulut Tasarım Kalıbı modelleri incelenmiştir. Bu modellere ön bellek yapısı oluşturulmuş ve yerleştirilmiştir. Modeller farklı programlama dilleri ve Hazelcast ön bellek teknolojisiyle geliştirilmiştir. Bu tasarım kalıplarının testleri tüm kayıtları çağıran ve tek bir kaydı çağıran metotlar üzerinden test edilmiş ve kıyaslanmıştır. Bu modellere alternatif bir model olarak Sepet Tasarım Kalıbı (Sidecar Pattern) sunulmaya çalışılmıştır. Sepet Tasarım Kalıbı, Gömülü Dağıtılmış Tasarım Kalıbı ve İstemci-Sunucu Kalıplarının ana bir konteyner yapısında tuttuğu, izleme (monitoring) için Docker bileşenini kullanabildiği bir yapıdır. Ön bellek ve uygulama konteynerları ayrıdır.

Sepet Tasarım Kalıbı modelinde Gömülü Dağıtılmış Tasarım Kalıbına benzer bir yapıda uygulama katmanı oluşturulmuştur. Uygulama katmanının aynı makinede olması ve kaynakları ön bellek ile paylaşması ortak özelliğidir. Bu yapıda Kubernetes Servisi kullanılarak diğer yapılara göre ölçeklendirilebilir bir yapı haline gelmiştir. Yük Dengeleyici'ye (Load Balancer) gelen isteklerin sayısını ve Yük Dengeleyici için ayrılacak kaynağın metriklerini bağımsız olarak ayarlayabiliriz. Gömülü Dağıtılmış Tasarım Kalıbına göre ek maliyet Kubernetes servisini kullanmak olacaktır.

Sepet Tasarım Kalıbının İstemci-Sunucu Tasarım Kalıbıyla ortak özelliği ise, ekosistemde yer alan servislerin, uygulamaların her birinin farklı bir programlama diliyle geliştirilebiliyor olmasıdır. Her bir uygulamaya farklı kaynak ayırmak mümkündür. Bu uygulamalar Kubernetes Pod'larında yer alır. Kubernetes bölmelerini uygulamanın kullanırlığına, aktifliğine ve büyüklüğüne göre ölçeklendirebiliriz.

6.1. Geçerliliğe Yönelik Şartlar ve Tehditler

Yapılan çalışmalarda şu durumlarda Sepet Tasarım Kalıbı modelinin avantajları olacaktır.

i. Java Sanal Makineleri olmayan bir yapıda CRM uygulaması Gömülü Dağıtılmış Tasarım Kalıbıyla birlikte entegre edilebilir.
ii. Kubernetes ile Hazelcast ön belleğe alma teknolojisinin birlikte çalışması gereken Java dili dışında geliştirilmiş bir uygulama için kullanılabilir.
iii. Hazelcast üyeleri arasında iletişim gerektiren fakat İstemci-Sunucu mimarisine sahip yapılar için bu model alternatif olarak sunulabilir.

Bu avantajların dışında dezavantaj oluşturduğu durumlar ortaya çıkmıştır. Dezavantajlı durumlar Gömülü Tasarım Kalıplarına ve İstemci-Sunucu Tasarım Kalıplarına göre farklılık oluşturmuştur.

i. Gömülü Tasarım Kalıbında 10 kullanıcı ile yapılan tek bir kaydı getiren metotun testi sonucunda yanıt verme süreleri Gömülü Dağıtılmış Tasarım Kalıbına göre daha azdır. Kullanıcı sayısının az olması ve tek bir kayıt getirilmesi istenildiği durumda ek olarak maliyet içereceği için Gömülü Tasarım Kalıbı daha uygun olacaktır. Diğer dezavantajlar ise şu şekildedir.
ii. Kubernetes bölme (pod) yapısının takibi ve servisin durumunun sürekli çalışır halde kalmasını sağlamak maliyet gerektirmiştir.
iii. Uygulamaların dockerize edilmesi diğer yöntemlerde olmayan ekstra bir maliyettir. Docker'da oluşturulan konteyner'ların ayakta kalması gerekir ve servislerin sayısına göre bu işlemlerin ayrı ayrı yapılması gerekmektedir.

Sonuçlar kısmında bu karşılaştırmalar üzerinden Sepet Tasarım Kalıbının entegre edilmesi ve test edilmesi sonrasında oluşan yanıt verme sürelerinden ve gecikmelerden bahsedilmiştir.

7. SONUÇLAR

Mikroservis ekosisteminde beş farklı modelde servislerin ön bellek yapısının kurgulanması, yönetilmesi ve performans karşılaştırılması yapılmıştır. Sonuçlar ve bulgular paylaşılmıştır. Toplamda 10, 100, 1000 kullanıcı için her modelde iki farklı metodun testleri yapılmış olup 18 farklı test sonucu ortaya çıkarılmıştır. Her mimari modelde aşağıdaki testler gerçekleştirilmiştir;

- MySQL ortamındaki veritabanında 1000 kayıtlık bir tabloda tutulan kayıtlar her modelde Hazelcast teknolojisiyle ön belleğe alınarak sorgulama işlemleri yapılmıştır.
- Test metotları ön bellekten tüm kayıtları çağırma ve tek bir kaydı çağırma işlemidir. Bu kıyaslamaların sonucunda modellerin birbirine göre ortalama yanıt süresi, gecikme süreleri ve bağlantı süreleri üzerinden karşılaştırma yapılmıştır.
- Ön bellek yapısının mikroservis ekosisteminde yer alan mimari yapılarda konumlandırılmasının performansa etkisi, bu modellerin kaynaklarının ve uygulamaların yapısının incelenmesi, ön belleğe alma politikası olarak kullanılan En Son Kullanılan Ön Bellek Yöntemi yapısının ortalama yanıt verme sürelerine etkisi test edilmiştir.

Gömülü Tasarım Kalıbı ve Gömülü Dağıtılmış Tasarım Kalıbı modellerindeki testlerde iki farklı metot için yanıt verme süreleri karşılaştırılmıştır. Ön belleğe alma tekniklerinin birbirlerine göre avantajı ve dezavantajı açısından kıyaslanmıştır. Gömülü Tasarım Kalıbı yapısında 1000 kullanıcıyla yapılan testte, 340 ve 397 numaralı kullanıcıların sistemden yanıt alma süreleri TCP bağlantısındaki gecikmeden kaynaklı ortalama sürenin iki katına çıkmıştır. Karşılaştırma sonuçlarında, Gömülü Dağıtılmış Tasarım Kalıbının 10 veya daha fazla kullanıcı ile çalışma yapıldığında yanıt verme sürelerinin daha az olduğu gözlemlenmiştir. Örnek olarak çalıştırılan CRM senaryosunda bu Gömülü Dağıtılmış Tasarım Kalıbının modeli önerilmektedir. Bu iki çalışmanın da dezavantajı gömülü sistemlerde çalıştığı için ölçeklendirme veya yedekleme gibi işlemlerin yapılamamasıdır.

İstemci-Sunucu Tasarım Kalıbı ve Bulut Tasarım Kalıbı modellerindeki karşılaştırmalarda tüm kayıtları çağırma ve tek bir kaydı çağırma üzerinden testler yapılmıştır. İstemci-Sunucu Tasarım Kalıbı ve Bulut Tasarım Kalıbının ortak avantajı

sunucu yapısının ölçeklendirilebilir, yedeklenebilir ve yönetilebilir bir yapıya sahip olmasıdır. Programlama diline bağlı olmaksızın servislerin farklı şekilde geliştirilebilmesi bu modellerin avantajlarından birisidir. Fakat bu iki modelde diğer yöntemlere göre daha fazla gecikme süreleri ortaya çıkmıştır. Bir diğer dezavantaj ise, sunucuların veya bulut ortamının gerektirdiği yapılandırmaların olmasıdır.

Sepet Tasarım Kalıbı (Sidecar Pattern) modelinin, konfigürasyon yapısı ve dağıtımı diğer modellere göre daha basittir. Uygulamaların dağıtımı Kubernetes pod yapısında olduğu için gecikme süreleri en aza indirgenmiştir. Verilerin ve uygulamaların izolasyonu Kubernetes ortamında sağlanmıştır. Sepet Tasarım Kalıbı yapısının ön belleğe alma işleminde ortaya çıkan sorun ise, konteyner tabanlı mimarilerin sınırlı olmasıdır. Konteyner mimarisinde ölçeklendirilebilir bir yapı mevcut değildir. Uygulamanın ve verinin konumları aynı pod'larda bulunduğu için yönetimi diğer modellere göre daha zordur.

Çalışmamızın sonucunda beş farklı modelin test sonuçları incelenmiş olup, karşılaştırmalar ve implementasyonlar sonucu oluşan bulgular doğrultusunda örnek verilen CRM uygulamasına ait senaryolarda Sepet Tasarım Kalıbı yönteminin uygulanabileceği durumlar ortaya çıkmıştır. Tez kapsamında Sepet Tasarım Kalıbının geçmişte kullanılan diğer modellere göre daha hızlı yanıt verme süresinin olduğu durumlar ortaya çıkarılmış olup yapılan test sonuçlarıyla doğrulanmıştır. Gelecekte yapılacak çalışmalar için Sepet Tasarım Kalıbının bir diğer varyasyonu olan henüz literatürde gelişmiş bir yapısı olmayan Ters Vekil Sepet Tasarım Kalıbı (Reverse Proxy Sidecar Pattern) adında mimari model için geliştirmeler yapılabilir.

KAYNAKÇA

[1] A. Akbulut and H. G. Perros, "Performance Analysis of Microservice Design Patterns," in IEEE Internet Computing, vol. 23, no. 6, pp. 19-27, 1 Nov.-Dec. 2019, doi: 10.1109/MIC.2019.2951094.

[2] A. O. Al-Abbasi and V. Aggarwal, "TTLCache: Taming Latency in Erasure-Coded Storage Through TTL Caching," in IEEE Transactions on Network and Service Management, vol. 17, no. 3, pp. 1582-1596, Sept. 2020, doi: 10.1109/TNSM.2020.2998175.

[3] A. Srivatsa, S. Nagel, N. Fasfous, N. Anh Vu Doan, T. Wild and A. Herkersdorf, "HyVE: A Hybrid Voting-based Eviction Policy for Caches," 2020 IEEE Nordic Circuits and Systems Conference (NorCAS), 2020, pp. 1-7, doi: 10.1109/NorCAS51424.2020.9265136.

[4] B. Nour, H. Khelifi, H. Moungla, R. Hussain and N. Guizani, "A Distributed Cache Placement Scheme for Large-Scale Information-Centric Networking," in IEEE Network, vol. 34, no. 6, pp. 126-132, November/December 2020, doi: 10.1109/MNET.011.2000081.

[5] Dattatreya Nadig, N. (2019). Testing Resilience of Envoy Service Proxy with Microservices.

[6] Fowler M., (2015, July 1). *Microservice Trade-Offs.* https://martinfowler.com/articles/microservice-trade-offs.html

[7] Gan, Y., & Delimitrou, C. (2018). The architectural implications of cloud microservices. IEEE Computer Architecture Letters, 17(2), 155-158.

[8] Hazelcast (2020). *Hazelcast Docs Overiew.* https://docs.hazelcast.com/imdg/3.12/hazelcast-overview.html.

[9] Hazelcast Documentation (2020). *Discovering Members by Multicast.* https://docs.hazelcast.com/imdg/4.2/clusters/discovering-by-multicast.html.

[10] J. Jenkins, G. Shipman, J. Mohd-Yusof, K. Barros, P. Carns and R. Ross, "A Case Study in Computational Caching Microservices for HPC," 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017, pp. 1309-1316, doi: 10.1109/IPDPSW.2017.40.

[11] J. Xiong, Y. Fang, P. Cheng, Z. Shi and W. Zhang, "Distributed Caching in Converged Networks: A Deep Reinforcement Learning Approach," in IEEE Transactions on Broadcasting, vol. 67, no. 1, pp. 201-211, March 2021, doi: 10.1109/TBC.2020.2996087.

[12] Leszko R, (2019, September 10). *Where is my cache? Architectural Patterns for Caching Microservices.* https://hazelcast.com/blog/architectural-patterns-for-caching-microservices/.

[13] Liu, H., Chen, S., Bao, Y., Yang, W., Chen, Y., Ding, W., & Shan, H. (2018, December). A High Performance, Scalable DNS Service for Very Large Scale Container Cloud Platforms. In Proceedings of the 19th International Middleware Conference Industry (pp. 39-45).

[14] L. J. Jagadeesan and V. B. Mendiratta, "When Failure is (Not) an Option: Reliability Models for Microservices Architectures," 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), 2020, pp. 19-24, doi: 10.1109/ISSREW51248.2020.00031.

[15] Loukides M., Swoyer S. (2020, July 15). *Microservices Adoption on 2020*. O'REILLY. https://www.oreilly.com/radar/microservices-adoption-in-2020/.

[16] M. Dehghan, L. Massoulié, D. Towsley, D. S. Menasché and Y. C. Tay, "A Utility Optimization Approach to Network Cache Design," in IEEE/ACM Transactions on Networking, vol. 27, no. 3, pp. 1013-1027, June 2019, doi: 10.1109/TNET.2019.2913677.

[17] N. C. Mendonca, C. M. Aderaldo, J. Camara and D. Garlan, "Model-Based Analysis of Microservice Resiliency Patterns," 2020 IEEE International Conference on Software Architecture (ICSA), 2020, pp. 114-124, doi: 10.1109/ICSA47634.2020.00019.

[18] Post, D. M., Snyder, M. V., Finck, E. J., & Saunders, D. K. (2006). Caching as a strategy for surviving periods of resource scarcity; a comparative study of two species of Neotoma. Functional Ecology, 20(4), 717-722.

[19] S. Borst, V. Gupta and A. Walid, "Distributed Caching Algorithms for Content Distribution Networks," 2010 Proceedings IEEE INFOCOM, 2010, pp. 1-9, doi: 10.1109/INFCOM.2010.5461964.

[20] S. Wen, D. Qin, T. Lv, L. Ge and X. Yang, "Traffic identification algorithm based on improved LRU," 2020 7th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2020 6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), 2020, pp. 157-159, doi: 10.1109/CSCloud-EdgeCom49738.2020.00034.

[21] Salhi, H., Odeh, F., Nasser, R., & Taweel, A. (2017, April). Open source in-memory data grid systems: Benchmarking hazelcast and infinispan. In Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering (pp. 163-164).

[22] W. Zhang, J. Xiong, L. Gui, B. Liu, M. Qiu and Z. Shi, "Distributed Caching Mechanism for Popular Services Distribution in Converged Overlay Networks," in IEEE Transactions on Broadcasting, vol. 66, no. 1, pp. 66-77, March 2020, doi: 10.1109/TBC.2019.2902818.

[23] Wang, W., Liu, Z., Jiang, Y., Yuan, X., & Wei, J. (2014, November). EasyCache: a transparent in-memory data caching approach for internetware. In Proceedings of the 6th Asia-Pacific Symposium on Internetware on Internetware (pp. 35-44).

[24] Weng Meizhen, Shang Yanlei and Tian Yue, "The design and implementation of LRU-based web cache," 2013 8th International Conference on Communications and Networking in China (CHINACOM), 2013, pp. 400-404, doi: 10.1109/ChinaCom.2013.6694629.

[25] Wu, N., Zuo, D., & Zhang, Z. (2018, November). An extensible fault tolerance testing framework for microservice-based cloud applications. In Proceedings of the 4th International Conference on Communication and Information Processing (pp. 38-42).

[26] Y. Yao, X. Kong, J. Zhou, X. Xu, W. Feng and Z. Liu, "An Advanced Adaptive Least Recently Used Buffer Management Algorithm for SSD," in IEEE Access, vol. 7, pp. 33494-33505, 2019, doi: 10.1109/ACCESS.2019.2904639.