KSÜ Mühendislik Bilimleri Dergisi, 28(3), 2025
Araştırma Makalesi

KSU J Eng Sci, 28(3), 2025
Research Article

Kahramanmaras Sutcu Imam University
Journal of Engineering Sciences

Geliş Tarihi : 24.02.2025
Kabul Tarihi : 21.05.2025

Received Date : 24.02.2025
Accepted Date : 21.05.2025

MİKROSERVİS YAKLAŞIMINDA SERVİSLER ARASI İLETİŞİM MİMARİLERİNİN PERFORMANS KARŞILAŞTIRMASI

PERFORMANCE COMPARISON OF INTER-SERVICE COMMUNICATION ARCHITECTURES IN MICROSERVICES APPROACH

Yalçın YELPAZE[1*] (ORCID: 0009-0006-0620-9667)
Serpil YILMAZ[2] (ORCID: 0000-0002-6276-6058)

[1] İzmir Katip Çelebi Üniversitesi, Yazılım Mühendisliği Bölümü, İzmir, Türkiye
[2] İzmir Katip Çelebi Üniversitesi, Bilgisayar Mühendisliği Bölümü, İzmir, Türkiye

*Sorumlu Yazar / Corresponding Author: Yalçın YELPAZE, yelpaze168@gmail.com

ÖZET
Mikroservis mimarisi, yazılım dünyasında giderek daha fazla benimsenen bir yaklaşım olup, servisler arası iletişim bu mimarinin en kritik unsurlarından biri olarak öne çıkmaktadır. Mikroservislerin birbirinden bağımsız olarak farklı programlama dilleri ve teknolojilerle geliştirilebilmesi, bu mimarinin popülaritesini artıran önemli bir faktördür. Ancak, bu tür bir yapı içinde servisler arasındaki iletişim süreçleri genellikle karmaşık olarak karşımıza çıkmakta ve servisler arasındaki iletişim yöntemlerinin performansları önemli hale gelmektedir. Bu zorlukları aşmak için çeşitli iletişim stratejileri ve mimariler geliştirilmiştir. Bu makalede, mikroservisler arasındaki iletişim yöntemlerinin performansları incelenmiştir. İş yükü, performans, güvenilirlik ve genel sistem mimarisi gibi kriterlere göre hangi iletişim yönteminin daha uygun olduğu, örnek vaka çalışmalarıyla yapılan bir değerlendirme ile tartışılmıştır.
Anahtar Kelimeler: Mikroservis, servisler arası iletişim, iletişim mimarileri.

ABSTRACT
Microservice architecture has become an increasingly adopted approach in software engineering, with inter-service communication emerging as one of its critical components. The ability to develop microservices independently using various programming languages and technologies is an important factor that increases the popularity of this architecture. However, in such a software development environment, communication processes between services can be complex, and the performance of communication methods between services becomes important. Various communication strategies and architectural solutions have been developed to overcome these challenges. This paper focuses on the performance of communication methods between microservices. The suitability of each communication method based on criteria such as workload, performance, reliability, and overall system architecture is discussed through an evaluation supported by case study examples.
Keywords: Microservice, inter-service communication, communication architectures.

GİRİŞ
Mikroservis mimarisi, büyük ve karmaşık yazılım projelerinin daha yönetilebilir ve ölçeklenebilir hale gelmesi için son yıllarda oldukça fazla tercih edilen bir yaklaşımdır. Ancak, bu mimarinin dağıtık yapısı, farklı servisler arasındaki iletişimi karmaşık hale getirebilmektedir. Bu noktada, servisler arası iletişim için kullanılan yöntemlerin etkinliği, sistem performansı, güvenilirlik ve ölçeklenebilirlik açısından kritik bir faktör olmaktadır.

Mikroservis mimarisinin amacı "düşük bağlılık (loosely coupled)" mikroservisler oluşturmaktır ve mikroservisler arası iletişim, bu amaç için en önemli yapıtaşlarından biridir (Bakshi, 2017). Geleneksel RESTful iletişim yöntemlerinin yanı sıra, sürekli değişen ve gelişen teknolojik ortamın ihtiyaçlarına cevap verebilmek için yeni yaklaşımlara olan gereksinim artmıştır. Bu bağlamda, RPC (Remote Procedure Call) ve mesaj kuyrukları gibi

ToCite: YELPAZE, Y., & YILMAZ, S., (2025). MİKROSERVİS YAKLAŞIMINDA SERVİSLER ARASI İLETİŞİM MİMARİLERİNİN PERFORMANS KARŞILAŞTIRMASI. *Kahramanmaraş Sütçü İmam Üniversitesi Mühendislik Bilimleri Dergisi, 28(3), 1246-1254*.

alternatif iletişim yöntemleri üzerinde kapsamlı araştırmalar yürütülmektedir (Hassan, 2024). Özmen vd. (2018), mikroservis mimarisinin avantajlarını ve karşılaşılan zorlukları incelemiştir. Çalışmada, mikroservislerin çeviklik, esneklik ve ölçeklenebilirlik gibi faydalar sağladığı; ancak, ağ üzerindeki hizmet keşfi, güvenlik yönetimi ve iletişim optimizasyonu gibi gereksinimlerin de dikkate alınması gerektiği vurgulanmıştır. Ayrıca, teknoloji şirketlerinin mikroservis mimarisi kapsamında karşılaştıkları faktörler ve bu faktörlere yönelik geliştirdikleri çözümler ele alınmıştır. Tapia vd. (2020), monolitik ve mikroservis mimarilerini performans açısından karşılaştırmalı olarak incelemiştir. Gerçekleştirilen testlerde, mikroservis mimarisinin işlem süresi, kaynak kullanımı ve yanıt süresi gibi metriklerde belirli senaryolarda monolitik yapılardan daha avantajlı olduğu görülmüştür. Ancak mikroservis geçiş sürecinde mimari karmaşıklık ve entegrasyon maliyetlerinin arttığı da vurgulanmıştır. Çalışma, performans kazancı için mimari geçişin dikkatle planlanması gerektiğini belirtmektedir. Herken vd. (2022), mikroservis mimarisinde asenkron iletişimin sistem verimliliğini artırdığını belirtmiştir. Çalışmada, asenkron iletişim sayesinde servislerin zaman bağımsız olarak veri alışverişi gerçekleştirebildiği ifade edilmiştir. Bu yapı, sistemin esnekliğini, hata toleransını ve ölçeklenebilirliğini artırmakta; mesaj kuyrukları aracılığıyla servisler arası bağımlılığı azaltmakta ve sistemin yüksek trafik altında dahi çalışmaya devam etmesini sağlamaktadır. Hataların izole edilerek diğer servisleri etkilememesi sayesinde, bu yaklaşımın dağıtık sistemlerde kararlılığı ve iş sürekliliğini desteklediği ortaya konmuştur. Blinowski vd. (2022), monolitik ve mikroservis mimarilerini performans ve ölçeklenebilirlik açısından karşılaştırmıştır. Gerçekleştirilen deneylerde, mikroservis mimarisinin özellikle yüksek trafik altında daha iyi ölçeklenebilirlik sağladığı; ancak düşük yük altında monolitik yapının daha stabil ve verimli çalıştığı gözlemlenmiştir. Mikroservisler, dağıtık yapıları sayesinde yatayda daha kolay genişletilebilmekte; ancak bu esneklik, daha yüksek kaynak tüketimiyle birlikte gelmektedir. Çalışmada, uygulama mimarisi tercihlerinin sistem gereksinimlerine göre yapılması gerektiği vurgulanmıştır. Servisler arası iletişim yöntemlerinin doğru bir şekilde seçilmesi, yalnızca performans ve güvenilirlik değil, aynı zamanda uygulama geliştirme maliyetlerini ve bakım süreçlerini de etkilemektedir. Bu nedenle, geliştiricilerin farklı yöntemlerin avantajlarını, dezavantajlarını ve hangi senaryolarda kullanılabileceklerini iyi analiz etmeleri gerekmektedir. Yelpaze vd. (2024), RESTful API, gRPC, mesaj kuyrukları, GraphQL ve WebSocket gibi iletişim yöntemlerinin; avantajları, dezavantajları ve hangi senaryolarda hangi iletişim mimarisinin tercih edilmesi gerektiğini incelemiştir.

Bu makalede, ilgili iletişim yöntemlerinin performanslarını karşılaştırmak amacıyla 3 adet benchmark çalışması gerçekleştirilmiş ve performans sonuçları değerlendirilmiştir. Elde edilen sonuçlara dayanarak hangi iletişim yönteminin hangi senaryolarda daha hızlı olduğuna dair önerilerde bulunulmuştur. Makalenin geri kalanı şu şekilde yapılandırılmıştır: Servisler Arası İletişim Mimarileri bölümünde, mikroservisler arasındaki iletişim yöntemleri tanıtılmış; her bir yöntemin avantaj ve dezavantajları açıklanmış ve ayrıca her iletişim türüne ilişkin örnek uygulamalara yer verilmiştir. Bunun yanı sıra, iletişim yöntemleri karşılaştırmalı olarak değerlendirilmiştir. Benchmark/Vaka Çalışmaları bölümünde, çeşitli vaka çalışmaları sunularak yöntemlerin uygulamadaki performansları analiz edilmiştir. Sonuçlar ve Tartışma bölümünde ise, elde edilen bulgular değerlendirilmiş ve çalışmanın genel sonuçları detaylı olarak aktarılmıştır.

SERVİSLER ARASI İLETİŞİM MİMARİLERİ

RESTful API

REST (Representational State Transfer), HTTP protokolü üzerine kurulu hafif bir iletişim yöntemidir. Bu iletişim yöntemi günümüzde en çok tercih edilen yöntemdir. Bunun en önemli sebebi, öğrenmesi kolay ve maliyetinin düşük olmasıdır. RESTful API'ler, genellikle JSON veya XML formatlarında veri alışverişi yapar ve aşağıdaki HTTP metotlarını kullanır:

- **GET**: Veri alma.
- **POST**: Yeni bir veri oluşturma/ekleme.
- **PUT**: Var olan veriyi güncelleme.
- **DELETE**: Veri silme.

RESTful API, web tabanlı uygulamalarda en yaygın kullanılan iletişim yöntemlerden biridir. Örneğin, bir e-ticaret uygulamasında ürünlerin listelenmesi, siparişlerin oluşturulması ve müşteri bilgilerine erişim RESTful API aracılığıyla kolayca gerçekleştirilebilir. İki farklı formatta iletişim kurulan bu yöntemde, JSON formatı XML formatına göre daha hızlı ve daha güvenlidir (Amazon Web Services. (t.y.-a). Bu nedenle REST protokolü genellikle JSON formatını kullanır (Weerasinghe & Perera, 2023).

Avantajları

- **Basitlik ve Esneklik**: RESTful API'ler, HTTP protokolüne dayandığı için basit ve anlaşılırdır.
- **Çapraz Platform Desteği**: REST, farklı teknolojiler ve platformlar arasında kolayca kullanılabilir.
- **Önbellekleme**: HTTP'nin cache mekanizmaları RESTful API'lerde kullanılabilir, bu da performansı artırabilir.

Dezavantajları

- **Performans Kısıtları**: RESTful API'ler metin tabanlı veri formatları kullandığı için bant genişliği kullanımı daha yüksektir.
- **Durumsuzluk**: Her istek bağımsız olarak işlenmek zorunda olduğundan, durumsuz yapı bazı karmaşıklıklara yol açabilir.
- **Zaman Aşımı Sorunları**: Uzun süreli işlemler için REST yetersiz kalabilir.
- **Desteklenen Yapı Türü Kısıtları**: JSON ile dosya transfer işlemleri yapılamamaktadır.

Örnek Uygulama: E-Ticaret Uygulaması

Bir e-ticaret platformunda ürün listeleme için RESTful API kullanılır. Örneğin, bir GET isteğiyle ürün kataloğuna erişilir, POST isteğiyle yeni bir sipariş oluşturulur. Ancak, stok durumu veya ödeme doğrulaması gibi gerçek zamanlı işlemler için RESTful API yeterli olmayabilir ve daha etkin bir çözüm gerekebilir.

RPC/gRPC

gRPC (Google Remote Procedure Call), Google tarafından geliştirilen, HTTP/2 protokolü ve Protobuf (Protocol Buffers) veri serileştirme formatına dayalı bir iletişim yöntemidir. Son yıllarda kullanımı oldukça artmıştır. Bunun en temel sebebi diğer tüm iletişim yöntemlerinden hızlı olmasıdır (Yelpaze, 2024). Daha çok iç servis iletişiminde kullanılır ve REST'e göre daha hızlı ve etkin bir performans sunar (Amazon Web Services. (t.y.-b).

gRPC'nin çalışma prensibi ; İstemci(Client) servisi(Server) çağırır, istemci tarafındaki gRPC kütüphanesi protokol template'ini kullanarak uzak prosedür çağrısı yapar ve ilgili çağrıyı HTTP2 üzerinden gönderir. Sunucu, isteği çözümler ve ilgili prosedür çağrısını protokol template'ini kullanarak devam ettirir. Proto dosya örneği aşağıdaki Şekil 1'de gösterilmiştir.

```
syntax = "proto3";
service Greeter {
    rpc SayHello (HelloRequest) returns (HelloReply);
}
message HelloRequest {
    string name = 1;
}
message HelloReply {
    string message = 1;
}
```

Şekil 1. Proto Dosya Örneği

Avantajları

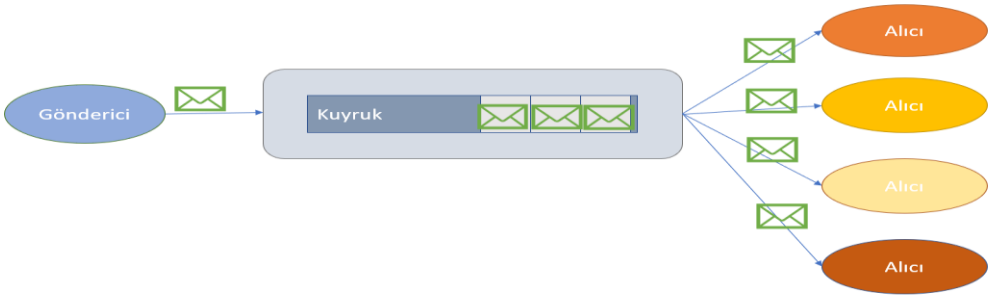
- **Yüksek Performans**: Protobuf formatı, JSON'dan daha kompakt ve hızlıdır.
- **Çift Yönlü Akış**: gRPC, istemci ve sunucu arasında çift yönlü veri akışını destekler.
- **Dil Bağımsızlığı**: gRPC, farklı programlama dilleri arasında sorunsuz çalışabilir.

Dezavantajları

- **Öğrenme Eğrisi**: REST'e göre daha karmaşık bir yapıya sahiptir.
- **HTTP/1.1 Uyumsuzluğu**: Sadece HTTP/2 protokolünü destekler.
- **Araç Desteği**: REST kadar geniş bir ekosisteme sahip değildir.

Örnek Uygulama: Gerçek Zamanlı Veri İşleme
Bir finansal uygulamada, hisse senedi fiyatlarının anlık güncellenmesi için gRPC kullanılabilir. Çift yönlü veri akışı sayesinde istemci ve sunucu arasında anlık iletişim sağlanır.

Mesaj Kuyrukları
Mesaj kuyrukları, asenkron iletişim sağlayarak servisler arasında bağımsızlık oluşturur. Mikroservisler arasındaki asenkron veri iletişimi için mesaj tabanlı haberleşme yöntemi kullanılır. RabbitMQ, Apache Kafka ve Amazon SQS gibi mesaj kuyruğu sistemleri, servislerin birbirleriyle etkileşimde bulunmasını sağlar. Bu yaklaşım, iş yükünü dengeli şekilde dağıtarak servislerin bağımsız çalışmasını kolaylaştırır.
Şekil 2, mesaj kuyruklarının çalışma sistemi bölümlerini göstermektedir. Gönderenler (Prodecure), mesajları kuyruğa (queue) yerleştirir ve alıcılar (receiver) bu mesajları kuyruktan alır. Kuyruk mekanizmasının kullanım amacı, mikroservisler veya sistem bileşenleri arasındaki veri ve mesaj akışını düzenlemektir.



Şekil 2. Mesaj Kuyrukları Çalışma Sistemi Bölümleri

Avantajları
- **Asenkron İletişim**: Servisler, mesajın iletilmesini beklemek zorunda kalmaz.
- **Yüksek Performans**: Büyük miktarda veri işleme kapasitesine sahiptir.
- **Güvenilirlik**: Mesajlar kuyrukta saklanır ve kaybolma riski minimize edilir.

Dezavantajları
- **Karmaşıklık**: Mesaj kuyruklarının yönetimi ve izlenmesi daha fazla efor gerektirir.
- **Gecikme**: Mesajların tesliminde küçük gecikmeler olabilir.

Örnek Uygulama: Sipariş İşleme Sistemi
Bir e-ticaret uygulamasında, siparişlerin işlenmesi için mesaj kuyrukları kullanılabilir. Örneğin; RabbitMQ, siparişlerin farklı servisler arasında dağıtımını sağlar ve işlemleri hızlandırır. E-ticaret üzerinden sipariş verildiğinde, siparişin e-posta bildirimi bazı durumlarda belirli bir süre sonra kullanıcıya ulaşır. Bunun sebebi, ilgili e-posta mesaj kuyruğuna iletilir ve buradan kullanıcıya gönderilir.

GraphQL
GraphQL, istemcilerin yalnızca ihtiyaç duyduğu veriyi sorgulamasına olanak tanıyan bir sorgulama dilidir. Eğer bir uygulama web, mobil ve masaüstü gibi farklı platformlarda çalışacaksa, GraphQL tek bir uç nokta üzerinden tüm istemcilerin veri gereksinimlerini karşılayabilir. Bu sayede, aynı veri kaynağından farklı istemcilerin değişen veri taleplerini esnek bir şekilde yönetmek mümkün olur.

Avantajları
- **Esnek Veri Sorgusu**: İstemciler yalnızca ihtiyaç duydukları alanları sorgular.
- **Tek Uç Nokta**: Tüm veriler için tek bir uç nokta sağlar.

Dezavantajları
- **Karmaşık Sunucu Yapısı**: Sunucunun doğru şekilde yapılandırılması zor olabilir.
- **Önbellekleme Zorluğu**: REST'e kıyasla önbellekleme daha karmaşıktır.

Örnek Uygulama: Sosyal Medya Uygulaması
Bir sosyal medya platformunda, kullanıcı profili ve gönderi bilgileri için GraphQL kullanılabilir. İstemci yalnızca görmek istediği alanları sorgular ve gereksiz veri aktarımı önlenir.

WebSocket
WebSocket, çift yönlü iletişimi destekleyen ve HTTP üzerinden başlatılan bir protokoldür. Modern web uygulamalarında gerçek zamanlı veri akışı ve asenkron iletişim için sıkça tercih edilir. Servisler, WebSocket aracılığıyla anlık veri paylaşımı yapabilir. Özellikle canlı sohbet sistemlerinde oldukça tercih edilmektedir. WebSocket, bir API teknolojisi olup, istemcinin sunucuya mesaj göndermesine ve sunucudan olay bazlı yanıtlar almasına olanak tanır. Bu süreç, sunucuya ek bir istek göndermeye veya yanıt beklemeye gerek kalmadan gerçekleşebilir.

Avantajları
- **Gerçek Zamanlı İletişim**: Veri anlık olarak iletilir.
- **Düşük Gecikme**: HTTP tabanlı çözümlere göre daha düşük gecikme süresi.

Dezavantajları
- **Durumsallık**: WebSocket bağlantılarının yönetimi daha karmaşıktır.
- **Güvenlik**: Sürekli açık bağlantılar ek güvenlik önlemleri gerektirir.

Örnek Uygulama: Canlı Sohbet Uygulaması
Bir canlı sohbet uygulamasında, kullanıcı mesajlarının anlık olarak iletilmesi için WebSocket kullanılabilir. Bu, düşük gecikme ve sürekli bağlantı avantajı sağlar.

İletişim Yöntemlerinin Karşılaştırılması
İletişim yöntemlerinin karşılaştırılması, mikroservisler arasında etkili ve verimli veri iletimi sağlamak için önemli bir adımdır. Tablo 1'de, yaygın kullanılan beş farklı mikroservis iletişim yöntemi olan RESTful API, gRPC, Mesaj Kuyrukları, GraphQL ve WebSocket teknolojileri çeşitli açılardan karşılaştırılmıştır.*

Performans açısından değerlendirildiğinde, gRPC, WebSocket ve mesaj kuyrukları yüksek performans sunarken, RESTful API nispeten düşük, GraphQL ise orta düzeyde performans göstermektedir. Veri formatı olarak RESTful API, JSON ve XML desteklerken; gRPC, verileri kompakt ve hızlı işlenen Protobuf formatında iletir. Mesaj kuyrukları mesaj tabanlı yapıya sahiptir, GraphQL yalnızca JSON desteklerken, WebSocket ise ikili veri iletimine olanak tanır. Çift yönlü veri akışı açısından, REST ve GraphQL bu özelliği desteklemezken; gRPC, mesaj kuyrukları ve WebSocket bu yeteneği destekleyerek daha dinamik veri iletişimi sağlar. Uyumluk açısından REST, HTTP/1.1 ve HTTP/2 ile çalışabilirken; gRPC yalnızca HTTP/2 ile uyumludur. Mesaj kuyrukları protokol bağımsız çalışabilirken; GraphQL HTTP tabanlıdır. WebSocket ise özel bir WebSocket protokolü kullanır.

Bu tablo, ilgili iletişim protokollerinin farklı sistem gereksinimlerine göre nasıl avantajlar sunduğunu ve hangi sınırlamalara sahip olduğunu açık bir şekilde ortaya koymaktadır.

BENCHMARK/VAKA ÇALIŞMALARI

Araştırma Yöntemi
Bu çalışma, Yin (2018) tarafından tanımlanan tekil gömülü vaka çalışması (single embedded case study) yaklaşımı doğrultusunda yapılandırılmıştır. Araştırmanın temel amacı, farklı mikroservis iletişim protokollerinin (REST, gRPC, GraphQL ve WebSockets) aynı koşullar altında nasıl performans gösterdiğini analiz etmektir. Yin'in vaka çalışması yaklaşımı, özellikle "nasıl" ve "neden" sorularının yanıtlandığı, araştırmacının olaylar üzerinde doğrudan bir kontrolünün bulunmadığı durumlar için uygun görülmektedir. Bu bağlamda, vaka çalışması yöntemi, farklı protokollerin performans farklarının bağlamsal olarak anlaşılması için tercih edilmiştir. Araştırmada, Yin'in önerdiği şekilde çoklu veri kaynakları kullanılmış ve veri toplama süreci sistematik biçimde yürütülmüştür. Performans verileri, farklı senaryolara dayalı olarak toplanmış ve analiz edilmiştir. Araştırma süreci boyunca sabit donanım/yazılım ortamı, tekrar eden testler ve tekil değişken kontrolü sağlanarak güvenilirlik artırılmıştır.

Tablo 1. İletişim Yöntemlerinin Karşılaştırılması

Özellik	RESTful API	gRPC	Mesaj Kuyrukları	GraphQL	WebSocket
Performans	Düşük	Yüksek	Yüksek	Orta	Yüksek
Veri Formatı	JSON, XML	Protobuf (ikili,binary format)	Mesaj tabanlı	JSON	İkili veri desteği sağlar.
Çift Yönlü Akış	Desteklenmez.	Desteklenir.	Desteklenir.	Desteklenmez.	Desteklenir.
Uyumluluk	HTTP/1.1 ve HTTP/2 desteği vardır.	Sadece HTTP/2 ile çalışır.	Protokol bağımsızdır.	HTTP tabanlıdır.	WebSocket protokolü kullanır.

*Bu Tablo, gRPC.io, RESTfulAPI.net, GraphQL.org ile Kafka/RabbitMQ'nun Resmi Dökümantasyonları Temel Alınarak Derlenmiştir.

Çalışma Ortamı

Vaka çalışmaları, Windows 10 Pro işletim sistemine sahip, 48 GB DDR4 RAM ve 2.80GHz hızında Intel(R) Core(TM) i7-1165G7 CPU barındıran bir fiziksel bilgisayarda gerçekleştirilmiştir. Veritabanı olarak MongoDB 5.0.5 sürümü kullanılmıştır. Her bir iletişim protokolü (REST, gRPC, GraphQL, WebSockets) için aynı işlevselliğe sahip dört ayrı mikroservis geliştirilmiştir. Bu mikroservisler, .NET 8.0 platformu kullanılarak yazılmıştır ve ortak bir JSON veri modeli kullanmaktadır: {id, ad, soyad, email}.Tüm mikroservisler ve veritabanı aynı fiziksel makinada çalıştırılmış, testler sırasında arka planda çalışan tüm diğer uygulamalar kapatılmış ve sistem kaynakları yalnızca testlere ayrılmıştır. Bu sayede dış etkilerden arındırılmış homojen bir test ortamı sağlanmıştır.

Veri toplama işlemi Postman uygulaması üzerinden manuel olarak gerçekleştirilmiş, her test üç kez ardışık olarak tekrarlanmıştır. Bu tekrarlar, ölçümlerin ortalama, minimum ve maksimum değerlerinin belirlenmesine olanak sağlamıştır.

Gerçekleştirilen Test Senaryoları

Üç farklı vaka çalışması gerçekleştirilmiştir: veritabanına(MongoDB 5.0.5) veri ekleme, veritabanından bir kayıt okuma ve veritabanından 100 kayıt okuma işlemleri. Performans değerlendirmesinde ortalama (mean), maksimum (max) ve minimum (min) değerler kullanılmıştır. Performans ölçüm sonuçları milisaniye (ms) cinsinden ölçülmüş ve raporlanmıştır.

Veritabanına veri ekleme

Veritabanına veri ekleme ile elde edilen test sonuçları Tablo 2'de gösterilmiştir. Tablo 2'de de görüldüğü üzere, gRPC iletişim yöntemi, veriyi en hızlı sürede veritabanına eklemiştir. İkinci sırada REST iletişim yöntemi gelmektedir. Üçüncü olarak WebSockets iletişim yöntemi düşük performans gösterirken, en düşük performansı ise GraphQL iletişim yöntemi göstermiştir.

Veritabanından bir kayıt okuma

İkinci test senaryosu olarak, veritabanından bir veri okuma işlemi gerçekleştirilerek performans testi yapılmıştır. Tek veri okuma işlemine ilişkin zaman ölçüm performansları Tablo 3'te sunulmuştur. REST ve gRPC protokolleri çok yakın sonuçlar göstermiş olsa da, gRPC protokolü en iyi performansı sergilemiştir. GraphQL protokolü en düşük performası göstermiştir. Websockets ise REST'ten daha yavaş, graphQL'den daha hızlı performans göstermiştir.

Veritabanından 100 adet kayıt okuma

Son olarak gerçekleştirilen bu test, veritabanından 100 öğe getirmenin protokol bazında ne kadar süre aldığını ölçmüştür. Veritabanında var olan 100 kayıt çekilmiştir. Herhangi bir filtreleme kullanılmamıştır. Elde edilen

sonuçlar Tablo 4'te sunulmuştur. gRPC en hızlı performansı gösterirken, WebSockets protokolü en yavaş performansı göstermiştir. REST protokolü ikinci en hızlı performas gösterirken, graphQL üçüncü sırada yer almıştır.

Tablo 2. Mikroservis İletişim Protokolleri Tek Veri Kaydetme Zaman Ölçüm Performanları

Mikroservis İletişim Protokolü	Performans Metrikleri	Değer (ms)
REST	Ortalama	2168
	En Düşük	1708
	En Yüksek	7669
gRPC	Ortalama	1402
	En Düşük	987
	En Yüksek	4712
GraphQL	Ortalama	4724
	En Düşük	4018
	En Yüksek	7041
WebSockets	Ortalama	2314
	En Düşük	1138
	En Yüksek	6522

Tablo 3. Mikroservis İletişim Protokolleri Tek Veri Okuma Zaman Ölçüm Performansları

Mikroservis İletişim Protokolü	Performans Metrikleri	Değer (ms)
REST	Ortalama	2179
	En Düşük	1727
	En Yüksek	5024
gRPC	Ortalama	1346
	En Düşük	1019
	En Yüksek	5001
GraphQL	Ortalama	6161
	En Düşük	4879
	En Yüksek	10989
WebSockets	Ortalama	2329
	En Düşük	1011
	En Yüksek	9381

Sonuçlar, elde edilen tablo verilerine dayanarak REST ve gRPC protokollerinin birbirine yakın performanslar sergilediğini, ancak en hızlı iletişim yönteminin gRPC olduğunu ortaya koymaktadır. Benzer şekilde, Berg vd. (2023) tarafından yapılan çalışmada da gRPC'nin üstün performansı gözlemlenmiştir. Bu durumun temel nedeni, gRPC'nin verileri binary (ikili) formatta iletmesidir. Çalışmamızda, REST'ten sonra en iyi performansı GraphQL protokolü göstermiştir; ancak bu bulgu, Jin vd. (2024) tarafından gerçekleştirilen ve farklı bir ortamda (AWS üzerinde) yürütülen çalışmayla çelişmektedir. Söz konusu farklılığın, altyapı farklarından kaynaklandığı düşünülmektedir. En düşük performans ise WebSocket protokolünde gözlemlenmiştir.

Tablo 4. Mikroservis İletişim Protokolleri 100 Adet Kayıt Okuma Zaman Ölçüm Performansları

Mikroservis İletişim Protokolü	Performans Metrikleri	Değer (ms)
REST	Ortalama	18718
	En Düşük	16748
	En Yüksek	37267
gRPC	Ortalama	10601
	En Düşük	9753
	En Yüksek	20957
GraphQL	Ortalama	21166
	En Düşük	9783
	En Yüksek	43518
WebSockets	Ortalama	22523
	En Düşük	20738
	En Yüksek	50920

TARTIŞMA VE SONUÇ

Bu makalede, mikroservis mimarileri ve servisler arası iletişim yöntemleri ayrıntılı bir şekilde incelenmiş, farklı mikroservis mimarilerinin performanları karşılaştırılmıştır. Ayrıca, her bir durum için hangi iletişim yöntemlerinin daha uygun olacağına dair öneriler sunulmuştur.

Literatürde yapılan önceki çalışmalar genellikle belirli iletişim protokollerini ikili biçimde karşılaştırmakla sınırlı kalmıştır. Bu çalışma ise, mikroservis mimarilerinde yaygın olarak kullanılan tüm temel iletişim yöntemlerini bir arada ele alarak performans açısından kapsamlı bir değerlendirme sunmaktadır. Böylece, farklı sistem ihtiyaçlarına göre hangi iletişim yönteminin daha uygun olabileceğine dair bütüncül bir bakış açısı sağlanmıştır.

Projelerde öncelikli olarak senkron iletişim yerine daha esnek ve ölçeklenebilir yapılar sunan asenkron iletişim yöntemlerinin tercih edilmesi önerilmektedir. Ancak, bazı senaryolarda performans açısından dezavantajlı olsa dahi senkron iletişim yöntemlerinin kullanılması gerekebilir. Örneğin, ödeme entegrasyonu içeren bir serviste, işlemlerin ödemeye bağlı olarak gerçekleştirilmesi gerektiğinden senkron iletişim kaçınılmazdır. Bu tür durumlarda, işlem güvenliği ve bütünlüğü açısından senkron yapı tercih edilmelidir. Öte yandan, anlık veri iletimi gerektiren uygulamalarda ise gerçek zamanlı iletişim desteği sunan WebSocket protokolü öne çıkmakta ve uygun bir çözüm olarak değerlendirilmektedir.

Eğer hız ön planda ve öğrenme maliyeti göz ardı edilebilecek bir faktörse, RPC tabanlı iletişim yöntemleri tercih edilmelidir. Ancak, RPC protokolü görece yeni ve hâlâ yaygınlık kazanmamış bir teknoloji olduğu için öğrenme süreci diğer protokollere göre daha yüksek maliyetli olabilir. Öte yandan, zaman kısıtlaması bulunan durumlarda, hem senkron hem de asenkron çalışma esnekliği sunan REST API yapısı uygun bir seçenek olarak öne çıkmaktadır. Mikroservis mimarisiyle tasarlanmış sistemlerde, servisler arası iletişim yöntemlerinin belirlenmesinde bu çalışmada sunulan değerlendirme kriterlerinin dikkate alınması; sistemlerin daha sağlam ve esnek bir altyapıya sahip olmasına, ayrıca gelecekteki kullanım gereksinimlerine daha uygun şekilde yapılandırılmasına katkı sağlayabilir.

KAYNAKLAR

Amazon Web Services. (t.y.-a). JSON ile XML arasındaki farklar nedir? https://aws.amazon.com/tr/compare/the-difference-between-json-xml/

Amazon Web Services. (t.y.-b). gRPC ile REST Arasındaki Fark Nedir? https://aws.amazon.com/tr/compare/the-difference-between-grpc-and-rest/

Bakshi, K. (2017). Microservices-based software architecture and approaches. IEEE Aerospace Conference, Big Sky, MT, USA, ss. 1-8. doi: 10.1109/AERO.2017.7943959.

Berg, J., & Redi, D. M. (2023). Benchmarking the request throughput of conventional API calls and gRPC: A comparative study of REST and gRPC (Bachelor's thesis, KTH Royal Institute of Technology). KTH DiVA Portal. https://kth.diva-portal.org/smash/get/diva2:1792957/FULLTEXT01.pdf

Blinowski, G., Ojdowska, A. and Przybyłek, A. (2022) Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation, in IEEE Access, vol. 10, pp. 20357-20374, doi: 10.1109/ACCESS.2022.3152803.

Hassan, Mohamed. (2024) Choosing the Right Communication Protocol for your Web Application. arXiv preprint arXiv:2409.07360.

Herken, Y. & Çambaşı, H. (2022). Mikroservis Mimarisi: Asenkron İletişim. YTE Blog. https://yteblog.bilgem.tubitak.gov.tr/mikroservis-mimarisi-asenkron-iletisim?

Jin, R., Cordingly, R., Zhao, D., & Lloyd, W. (2024). GraphQL vs. REST: A performance and cost investigation for serverless applications. In Proceedings of the 13th Workshop on Middleware and Applications (pp. 37–42). Association for Computing Machinery. https://doi.org/10.1145/3702634.3702956

Kami'nski, L., Kozlowski, M., Sporysz, D., Wolska, K., Zaniewski, P., & Roszczyk, R. (2022). Comparative Review of Selected Internet Communication Protocols. Foundations of Computing and Decision Sciences, 48, 39 – 56.

Özmen, N., & Tokdemir, G. (2018). Mikroservis mimarisi ve mimari faktörleri üzerine endüstriyel bir inceleme. In UYMS-YMO 2018 CEUR Workshop Proceedings. https://ceur-ws.org/Vol-2201/UYMS_YMO_2018_paper_106.pdf

Tapia, F., Mora, M. Á., Fuertes, W., Aules, H., Flores, E., & Toulkeridis, T. (2020). From Monolithic Systems to Microservices: A Comparative Study of Performance. Applied Sciences, 10(17), 5797. https://doi.org/10.3390/app10175797

Yelpaze, Y. (2024). Mikroservis Yaklaşımında Servisler Arası İletişim Mimarileri (Yüksek Lisans Tezi) İzmir Katip Çelebi Üniversitesi, Fen Bilimleri Enstitüsü, Yazılım Mühendisliği Anabilim Dalı, İzmir.

Yelpaze,Y., & Yılmaz S. (2024). Inter-Service Communication Architectures in Microservices Approach. ASES VIII. International Health, Engineering and Sciences Conference, Conference Book April 06-07 2024, İzmir, Türkiye.

Weerasinghe, S., & Perera, I. (2023). Optimized Strategy for Inter-Service Communication in Microservices. International Journal of Advanced Computer Science and Applications, 14(2).

Yin, R. K. (2018). Case study research and applications: Design and methods (6th ed.). SAGE Publications.