

MÜHENDİSLİK FAKÜLTESİ

YAZILIM MÜHENDİSLİĞİ YÜKSEK LİSANS

TEZSİZ YÜKSEK LİSANS DÖNEM PROJESİ

C# VE JAVA DİLLERİNDE ASYNC PROGRAMLAMANIN PERFORMANSA OLAN ETKİSİNİN İNCELENMESİ

HAZIRLAYAN

Türhan Yıldırım

DANIŞMAN ÖĞRETİM ÜYESİ

Dr. Öğr. Üyesi Bilgin Avenoğlu

2022

# ETİK İLKELERE UYGUNLUK BEYANI

Dönem proje yazma sürecinde bilimsel ve etik ilkelere uyduğumu, yararlandığım tüm kaynakları kaynak gösterme ilkelerine uygun olarak kaynakçada belirttiğimi ve bu bölümler dışındaki tüm ifadelerin şahsıma ait olduğunu beyan ederim.

İmza Adı Soyadı

**C# VE JAVA DİLLERİNDE ASYNC PROGRAMALAMANIN PERFORMANSA OLAN ETKİSİNİN İNCELENMESİ**

**Türhan Yıldırım**

**AHMET YESEVİ ÜNİVERSİTESİ**

**YAZILIM MÜHENDİSLİĞİ YÜKSEK LİSANS BÖLÜMÜ**

**2022**

# ÖZET

C# ve Java uygulamalarında dosya okuma yazma işlemleri için asenkron programlara performans kazancı sağlamamakla birlikte uygulamaların daha yavaş çalışmasına da sebep olmakta.

Asenkron programlamanın işlemci gücüne dayanan hesaplama içerek kısımları için ise C# için performans artışı sağlamaktadır.

Fakat hesaplama işlemleri içeren Java uygulamaları için aynı sonuç ortaya çıkmamıştır. Runnable, FutureTask, ExecutorService asenkron implementasyonu ile geliştirilen işlemler daha yavaş çalışmaktadır.

**Anahtar Kelimeler:** Sync, Async, Programalama, Performans

**Danışman:** Dr. Öğr. Üyesi Bilgin Avenoğlu

**DÖNEM PROJESİNİN İNGİLİZCE ADI BU BÖLÜME YAZILMALIDIR**

**Türhan Yıldırım**

**AHMET YESEVI UNIVERSITY**

**SOFTWARE ENGINEERING MASTER’S DEPARTMENT**

**2022**

# ABSTRACT

While it does not provide performance gain to asynchronous programs for file reading and writing operations in C # and Java applications, it also causes applications to run slower.

It provides a performance increase for C# for the computational parts of asynchronous programming based on processing power.

However, the same result did not occur for Java applications that contain computational operations. Operations developed with asynchronous implementations of Runnable, FutureTask, ExecutorService run slower.

**Keywords:** Sync, Async, Programming, Performance

**Advisor:** Dr. Öğr. Üyesi Bilgin Avenoğlu

# İÇİNDEKİLER

[ETİK İLKELERE UYGUNLUK BEYANI ii](#_Toc511318601)

[ÖZET iii](#_Toc511318602)

[ABSTRACT iv](#_Toc511318603)

[İÇİNDEKİLER v](#_Toc511318604)

[ŞEKİLLER LİSTESİ vi](#_Toc511318605)

[TABLOLAR LİSTESİ vii](#_Toc511318606)

[SİMGELER VE KISALTMALAR viii](#_Toc511318607)

[BÖLÜM I GİRİŞ 1](#_Toc511318608)

[1.1. Problem 1](#_Toc511318609)

[1.2. Araştırmanın Amacı 1](#_Toc511318610)

[1.3. Araştırmanın Önemi 1](#_Toc511318611)

[1.4. Sayıltılar 1](#_Toc511318612)

[1.5. Sınırlılıklar 1](#_Toc511318613)

[1.6. Tanımlar 1](#_Toc511318614)

[BÖLÜM II KAVRAMSAL ÇERÇEVE 2](#_Toc511318615)

[2.1. Literatür Taraması 2](#_Toc511318616)

[BÖLÜM III YÖNTEM 3](#_Toc511318619)

[3.1. Araştırmanın Modeli 3](#_Toc511318620)

[3.2. Evren ve Örneklem 3](#_Toc511318621)

[3.3. Veri Toplama Araçları 3](#_Toc511318622)

[3.4. Verilerin Toplanması 4](#_Toc511318623)

[3.5. Verilerin Analizi 4](#_Toc511318624)

[BÖLÜM IV BULGULAR VE YORUM 5](#_Toc511318625)

[4.1. Araştırma sorusuna(alt problem) ilişkin bulgular. 5](#_Toc511318626)

[BÖLÜM V SONUÇ, TARTIŞMA VE ÖNERİLER 6](#_Toc511318629)

[5.1. Sonuç 6](#_Toc511318630)

[5.2. Tartışma 6](#_Toc511318631)

[5.3. Öneriler 6](#_Toc511318632)

[KAYNAKÇA 7](#_Toc511318633)

[EKLER 8](#_Toc511318634)

# 

# ŞEKİLLER LİSTESİ

# TABLOLAR LİSTESİ

# SİMGELER VE KISALTMALAR

# BÖLÜM I **GİRİŞ**

## Problem

Günümüzde trendi olan asenkron(async) yazılım geliştirme mantığının popüler yazılım geliştirme dillerinden olan C# ve Java’da işlemci yükü gerektiren ve dosya okuma-yazma (File IO) işlemlerine olan etkisi gerçekten ifade edildiği gibi performans artışı sağlamakta mıdır?

## Araştırmanın Amacı

Asenkron programlama ile vaat edilen performans artışı işlem türüne göre farklılık gösteriyor mu tespit etmek istiyoruz. İşlem gücü (CPU) gerektiren ve dosya okuma-yazma (File IO) işlemi yapan işlemler karşılaştırılacaktır. Buna göre bu 2 koşul için performans etkileri incelenecektir.

## Araştırmanın Önemi

Gerçek bir performans artışı sağlanabilecek mi bilinmeden geliştirilen yazılım ilk günden asenkron(async) olarak yazılmaya çalışılmakta ve bu durum geliştirme sürecin de yazılım geliştirme sürenin, kompleksitesinin artmasına sebep olmaktadır.

## Sayıltılar

Herhangi bir sayıltı bulunmamaktadır.

## 1.5. Sınırlılıklar

İşlemci yükü oluşturan ve dosya okuma-yazma işlemleri için asenkron yazılım geliştirme metodolojisini kullanan tüm yazılımlar için geçerli bir durumdur.

## Tanımlar

Aynı işlemi defalarca kez tekrar eden ya da çok fazla işlem yükü olan uygulamalar/yazlımlar için bu durum geçerlidir. Tek kullanıcısı olan, zaman sınırı olmayan yani işlemin ne kadar süre de ne kadar işlemci yükü ile yapıldığının önemi olmayan yazılımlar için geçerli değildir.

# BÖLÜM II KAVRAMSAL ÇERÇEVE

## 2.1. Literatür Taraması

Yapılan literatür taramalarına göre asenkron yazılım geliştirme yaklaşımı üzerine oldukça fazla yazı/makale bulunmasına rağmen, var olan araştırmalar asenkron yazılım geliştirme yaklaşımının içsel çalışma mekanizmaları ve algoritmaları üzerinde yoğunlaşmaktadır. Ayrıca doğru bir asenkron yazılım parçasının nasıl yazılması gerektiği üzerinde durmaktadır. Asenkron olarak geliştirilen uygulamaların performans etkisinin analizine odaklanan bir araştırma bulunmamaktadır.

Genel literatür asenkron programlama modelleri ve uygulanışları üzerine yoğunlaşırken, linkte belirttiğim araştırma doğru bir şekilde geliştirilen asenkron uygulamalar için %30,7 gibi çok ciddi bir performans kazancı olabileceğini bildirmektedir. Tabi ki bu performans artışını diğer yayımlarda olduğu gibi işlemci yükü gerektiren ya da dosya okuma-yazma işlemi için olan asenkron uygulamalar özelinde araştırılmadığını asenkron uygulama geliştirmeye uygun bir senaryo için yapılan doğru bir asenkron yazılım geliştirme metodolojisinin uygulanması sonucu elde edilen performans değişiminin sonucu olduğunu bilmek gerekmektedir. (Castillo et al., 2019)

Asenkron işlem mantığına istinaden yayınlanan başka bir yayımda ise, asenkron programlama algoritmalarının senkronize işlem algoritmalarına göre 1,5-12 kat arasında performans kazancı sağladığını göstermektedir. İlgili yayımda FSSP adı verilen asenkron işlem gerçekleştirme algoritmasının sonuçlarını göstermektedir. İlgili yayımda işlemci gücü gerektiren yazılımlar ile dosya okuma-yazma üzerinde yoğunlaşan uygulamalar ayrımı yapılmaksınız performans yorumu yapılmaktadır. (Shi, Zhao, Zhang, Yoshigoe, & Chang, 2019)

Java ve X10 yazılım geliştirme dillerini hedef alıp bu dillerde geliştirilen asenkron uygulamalarda  
oluşabilecek “race condition” yani aynı anda erişilmeye çalışılan bir kaynağın olması ve kaynağın aslında aynı anda sadece bir çağrı cevap verebilmesi sonucu diğer çağrıların kilitlenip sırada kalması ve dar boğaz yaratması durumudur. Bu durum yani makalede anlatılan sonuç aslında bizimde bu tez kapsamında ölçemeye çalıştığımız bir dosyanın okunup-yazılması (IO) işlemine atıfta bulunmatdatır. Çünkü bir metin dosyasının içeriğinin okunma sürecince dosya yazılımsal olarak açılıp ve ilgili dosya içeriği satır satır okunur. Bu okuma işlemi yazılımsal olarak aynı anda sadece bir tane okuma istediğine cevap verebileceği için aslında bizim test senaryomuzda race condition sebep olması muhtemeldir. İlgili yayımda da bu tarz durumlar da asenkron olarak geliştirilen uygulamanın performans artışı yaşaması yerinde yaklaşık olarak 3.05 kat kadar uygulamanın senkron uygulamaya göre yavaş çalıştığı tespit edilmektedir.(Raman, Zhao, Sarkar, Vechev, & Yahav, 2010)

# BÖLÜM III **YÖNTEM**

## 3.1. Araştırmanın Modeli

Çalışmanın yapılacağı bilgisayar bilgileri

* AMD Ryzen 3900X (12 Fiziksel Çekirdek) İşlemci
* 32 GB 3200MHZ DDR4 Ram
* Samsung 970Evo Plus 512GB SSD Sabit Disk
* Windows 11 (21H2 – OS Build 22000.613) İşletim sistemi

C# için geliştirilecek uygulama ile ilgili bilgiler

* Visual Studio 2022 (17.1.5)
* .NET 6.0
* .NET SDK Version 6.0.202
* Console uygulaması yazılacak
* Asenkron kısmında ilgili methodlar Task.Run şeklinde çağrılacaktır

Java uygulaması için kullanılacak uygulama bilgileri

* IntelliJ IDEA 2022.1.1 (Community Edition) Build #IC-221.5591.52
* jre1.8.0\_331
* Console uygulaması yazılacak
* Asenkron kısımda ilgili methodlar için Runnable task şeklinde çağrılacaktır

## 3.2. Evren ve Örneklem

Veri toplama methodlarından elde edilecek sonuçlar kullanılacaktır.

## 3.3. Veri Toplama Araçları

Verinin toplanması için 2 temel method yazılacak.

* İşlemci yükü gereken method, içeriğinde 1’den 100’e kadar olan sayıların karelerini hesaplayacak
* Dosya okuma işlemi yapacak olan method, içeriğinde 1MB metinsel veri içeren (txt) dosyasını okuyacak

Bu iki temel method için C# ve Java dillerinde, ilgili methodları 1000’er kez senkron ve asenkron çağıran methodlar hazırlanacak. Bu işlem her bir durum için 10’ar kez tekrarlanacaktır.

* Senkron bir şekilde 1’den 10000’e kadar olan sayıların karelerini hesaplayan method 1 dakika boyunca çağrılacaktır ve işlemin kaç milisaniyede tamamlandığı alınacaktır.
* Senkron bir şekilde 1MB metinsel veri içeren txt dosya okunacak ve işlemin kaç milisaniyede tamamlandığı alınacaktır.
* Asenkron bir şekilde 1’den 100’e kadar olan sayıların karelerini hesaplayan method 1 dakika boyunca çağrılacaktır ve işlemin kaç milisaniyede tamamlandığı alınacaktır.
* Asenkron bir şekilde 1MB metinsel veri içeren txt dosya okunacak ve işlemin kaç milisaniyede tamamlandığı alınacaktır.

Geliştirilen uygulamalara aşağıdaki adreslerden ulaşabilirisiniz

* C# Uygulaması: <https://github.com/turhany/ThesisApps/tree/main/CSharp>
* Java Uygulaması: <https://github.com/turhany/ThesisApps/tree/main/Java>

## 3.4. Verilerin Toplanması

C# ve Java için geliştirilen uygulamalar tamamlanıp çalıştırılmış ve veri analizi bölümündeki çıktılar elde edilmiştir.

## 3.5. Verilerin Analizi

C# uygulaması için elde edilen veriler alttadır

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Dosya Okuma (milisaniye) | | Hesaplama (milisaniye) | |
| Deneme | Senkron | Asenkron | Senkron | Asenkron |
| 1 | 1294 | 1274 | 238 | 113 |
| 2 | 1044 | 1470 | 238 | 14 |
| 3 | 1060 | 1388 | 238 | 14 |
| 4 | 1047 | 1373 | 238 | 14 |
| 5 | 1060 | 1346 | 238 | 14 |
| 6 | 1038 | 1400 | 238 | 14 |
| 7 | 1041 | 1418 | 238 | 19 |
| 8 | 1432 | 1308 | 238 | 16 |
| 9 | 1680 | 1382 | 238 | 16 |
| 10 | 1665 | 1391 | 238 | 14 |
| Ortalama | 1236.1 | 1385 | 238.1 | 24.8 |

Java uygulaması için elde edilen veriler alttadır

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Dosya Okuma (milisaniye) | | Hesaplama (milisaniye) | |
| Deneme | Senkron | Asenkron | Senkron | Asenkron |
| 1 | 1050 | 875 | 1 (1598400 ns) | 37 |
| 2 | 862 | 694 | 0 (22100 ns) | 3 |
| 3 | 659 | 652 | 0 (20300 ns) | 2 |
| 4 | 653 | 757 | 0 (21800 ns) | 2 |
| 5 | 651 | 739 | 0 (21900 ns) | 3 |
| 6 | 646 | 928 | 0 (21800 ns) | 2 |
| 7 | 645 | 658 | 0 (21800 ns) | 2 |
| 8 | 644 | 699 | 0 (21800 ns) | 1 |
| 9 | 643 | 814 | 0 (21600 ns) | 2 |
| 10 | 644 | 497 | 0 (21800 ns) | 1 |
| Ortalama | 710 | 731 | 0 (179330 ns) | 6 |

# BÖLÜM IV BULGULAR VE YORUM

## 4.1. Araştırma sorusuna(alt problem) ilişkin bulgular.

Hazırlanılan C# ve Java uygulamasının çıktıları incelendiğinde aşağıdaki sonuçlara ulaşılmıştır.

C# uygulama özelinde incelendiğinde hesaplama methodu senkron işlem akışında görevini ortalama olarak 238.1 milisaniyede tamamlamıştır. Aynı işlem asenkron akış ile gerçekleştirildiğinde ise tüm işlemler ortalama 24.8 milisaniye sürmüştür. Buda c# uygulamasının hesaplama adımı için asenkron programlamanın yaklaşık 9.8 kat daha hızlı çalışmasını sağlamıştır. Elde edilen bu bilgiye göre işlemci yükü gerektiren işlemler için c# yazılım dilinde asenkron programlamanın performans artışı sağladığı saptanmıştır.

Yine C# uygulamasının dosya okuma akışı verileri incelendiğinde senkron akış ortalama 1236.1 milisaniye sürerken asenkron akış ise ortalama 1385 milisaniye sürmüştür. Bu verilere göre ise dosya işlemleri için asenkron akışın uygulanmasının performans artışına bir etkisinin olmadığı gibi yaklaşık %12 gibi bir performans kaybına da sebep olmuştur.

Java uygulamasının verileri incelendiğinde ise dosya işlemleri için C# uygulaması çıktıları ile aynı durum görülürken, hesaplama işlemi için ise tersine bir durum oluştuğu görülmektedir.

Java uygulamasının dosya okuma kısmı için senkron akış ortalama olarak 710 milisaniye sürmektedir. Asenkron akış ise 731 milisaniye sürmektedir. Bu Java yazılım dilinde asenkron dosya okuma işlemi yapan uygulamalar için senkron akışa göre yaklaşım %2,9 oranında bir yavaşlama demek olmaktadır.

Geliştirilen Java test uygulamasının işlemciye yüklenen hesaplama kısmı için ise senkron akışın ortalaması nano saniye(179330ns – 0.17ms) seviyelerinde olurken senkron akışın ortalaması ise 6 milisaniye olduğu görülmektedir. Buna göre geliştirilen asenkron uygulaması senkron uygulamaya göre yaklaşık 6 kat yavaş olduğu tespit edilmiştir.

# BÖLÜM V SONUÇ, TARTIŞMA VE ÖNERİLER

## 5.1. Sonuç

Toplanan test verilene göre c# ve Java uygulamalarında dosya okuma yazma işlemleri için asenkron programlama performans kazancı sağlamamakla birlikte uygulamaların daha yavaş çalışmasınada sebep olmakta.

Asenkron programlamanın işlemci gücüne dayanan hesaplama içerek kısımları için ise c# için çalışmamız özelinde 9.8 kat performans artışı sağlaması ile bu durumlar için net bir şekilde pozitif performans etkisi olduğu tespit edilmiştir.

Fakat Java hesaplama işlemleri içeren Java uygulamaları için aynı sonuç ortaya çıkmamıştır. Asenkron şeklinde yazılan işlemler 6 kattan fazla yavaş çalışmaktadır.

## 5.2. Tartışma

Araştırmasını yaptığımız asenkron uygulama performansı konusunda Java programlama dili içinde birçok farklı uygulama şekli bulunmaktadır. Araştımamız içinde java dilindeki “Runnable” ve “FutureTask” task implementastonu üzerinden “ExecutorService” asenkron task yönetimi akışı kullanılmıştır.

Bu uygulama geliştirme şekli dışında olan thread, CompletableFuture, ListenableFuture, EA-Async library, Oracle Fork/Join yöntemleri ile denemeler yapılmamıştır.

## 5.3. Öneriler

Tartışma kısmında da belirtildiği gibi java yazılım dili için diğer asenkron uygulama geliştirme yöntemleri ile de araştırma yapılabilir.

Bu yöntemlere örnekler; thread, CompletableFuture, ListenableFuture, EA-Async library, Oracle Fork/Join şeklindedir.

# KAYNAKÇA

Castillo, E., Jain, N., Casas, M., Moreto, M., Schulz, M., Beivide, R., … Bhatele, A. (2019). *Optimizing computation-communication overlap in asynchronous task-based programs*. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP*. doi:10.1145/3293883.3295720

Raman, R., Zhao, J., Sarkar, V., Vechev, M., & Yahav, E. (2010). *Efficient data race detection for async-finish parallelism*. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 6418 LNCS, pp. 368–383). doi:10.1007/978-3-642-16612-9\_28

Shi, H., Zhao, Y., Zhang, B., Yoshigoe, K., & Chang, F. (2019). Effective Parallel Computing via a Free Stale Synchronous Parallel Strategy. *IEEE Access*, *7*. doi:10.1109/ACCESS.2019.2936820

# EKLER

## Graphical user interface, text, application Description automatically generatedC# Uygulama Kodları

Graphical user interface, text, application

Description automatically generatedSenkron (Sync) Akış Uygulama Kodları

Graphical user interface, application

Description automatically generatedAsenkron (Async) Akış Uygulama Kodları

Ana İşlem Methodlarının Kodları

Graphical user interface, text, application

Description automatically generated

Örnek Test Çıktısı

Graphical user interface, text

Description automatically generated with medium confidence

## Text Description automatically generatedJava Uygulama Kodları

Text

Description automatically generatedSenkron (Sync) Akış Uygulama Kodları

Text

Description automatically generatedAsenkron (Async) Akış Uygulama Kodları

Text

Description automatically generated

Text

Description automatically generatedAna İşlem Methodlarının Kodları

Graphical user interface, text

Description automatically generatedÖrnek Test Çıktısı

Text

Description automatically generated