T.C.
BEYKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
MATEMATİK BİLGİSAYAR ANABİLİM DALI
BİLGİ TEKNOLOJİLERİ BİLİM DALI

YOĞUN YÜKTE ÇALIŞAN SİSTEMLERDE YAZILIMSAL OLARAK DAVRANIŞ FARKLILIKLARININ ANALİZ EDİLMESİ

Yüksek Lisans Tezi

Tezi Hazırlayan:

Melih SAKARYA

İSTANBUL, 2016

T.C.
BEYKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
MATEMATİK BİLGİSAYAR ANABİLİM DALI
BİLGİ TEKNOLOJİLERİ BİLİM DALI

YOĞUN YÜKTE ÇALIŞAN SİSTEMLERDE YAZILIMSAL OLARAK DAVRANIŞ FARKLILIKLARININ ANALİZ EDİLMESİ

Yüksek Lisans Tezi

Tezi Hazırlayan:

Melih SAKARYA

Öğrenci No:

130862021

Danışman:

Yrd. Doç. Dr. Turhan KARAGÜLER

İSTANBUL, 2016

YEMİN METNİ

Yüksek lisans tezi olarak sunduğum "**Yoğun Yükte Çalışan Sistemlerde Yazılımsal Olarak Davranış Farklılıklarının Analiz Edilmesi**" başlıklı çalışmanın, bilimsel ahlak ve geleneklere uygun bir şekilde tarafımdan yazıldığını, yararlandığım eserlerin tamamının kaynaklarda gösterildiğini ve çalışmanın içinde kullanıldıkları her yerde atıf yapıldığını belirtir ve bunu onurumla doğrularım. 12/01/2016

Aday: **Melih SAKARYA**

T.C.
BEYKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZ SAVUNMA SINAVI SONUÇ TUTANAĞI

Beykent Üniversitesi
Fen Bilimleri Enstitüsü Müdürlüğü'ne,

Aşağıda tez adı belirtilen yüksek lisans öğrencisi 130862026 no'lu MELİH SAKARYA'nın tarihinde yapılan tez savunma sınavı[1] sonucunda 50 dakika süreyle sunduğu ve savunduğu tezi hakkında[2] oybirliğiyle, KABUL kararı verilmiştir.

Bilgilerinize saygılarımızla arz ederiz.

Anabilim Dalı : BİLGİSAYAR MÜHENDİSLİĞİ
Programı : BİLGİSAYAR MÜHENDİSLİĞİ
Tez Başlığı[3] : Yapay Yükte Çalışan Sistemlerde Yapısal Olarak Devrans Fazlılıkların Analiz Edilmesi

Tez Sınav Jürisi	Öğretim Üyesi	İmza
Danışman :	Yrd. Doç. Dr. Turhan Kuraşiler	
Üye :	Yrd. Doç. Dr. R. Haluk Kul	
Üye :	Yrd. Doç. Dr. Ediz Saykol	

---

[1] Jüri üyeleri söz konusu tezin kendilerine teslim edildiği tarihten itibaren en geç bir ay içinde toplanarak öğrenciyi tez savunma sınavına alır. Belirlenen günde yapılamayan jüri toplantısı, katılanların hazırladığı bir tutanakla enstitü yönetimine bildirilir. Bu durumda jüri en geç onbeş gün içinde toplanarak adayı tez savunma sınavına alır. Tez savunma sınav süresi en az 45 dakikadır. Yüksek lisans tez savunma sınavı, tez çalışmasının sunulması ve bunu izleyen soru-yanıt bölümlerinden oluşur ve dinleyiciye açıktır. (Beykent Lisansüstü eğitim ve Öğretim Yönetmeliği-Madde30-3)

[2] Tez sınavının tamamlanmasından sonra jüri, tez hakkında "kabul", "düzeltme" veya "red" kararı verir. Jüri başkanı, jüri üyelerince imzalanmış sınav tutanağını, tez sınavını izleyen üç gün içinde ilgili enstitü yönetimine teslim eder. Tezi başarısız bulunan öğrencinin Enstitü ile ilişiği kesilir. Tezi hakkında düzeltme kararı verilen öğrenci en geç üç ay içinde gerekli düzeltmeleri yaparak ve yönetmelikte belirtilen usullere uygun olarak tezini aynı jüri önünde yeniden savunur. Bu savunma sınavında da tezi kabul edilmeyen öğrencinin enstitü ile ilişiği kesilir.(Beykent Lisansüstü eğitim ve Öğretim Yönetmeliği-Madde30-4)

[3] İleridedoğabilecekaksaklıklarınengellenmesiiçintezinbaşlığınınyazılmasıgerekmektedir.

YOĞUN YÜKTE ÇALIŞAN SİSTEMLERDE YAZILIMSAL OLARAK DAVRANIŞ FARKLILIKLARININ ANALİZ EDİLMESİ

Tezi Hazırlayan: **Melih SAKARYA**

ÖZET

Bilişim sistemlerinde artan iş yükünü yönetebilmek için uygulamalarda eksik olan, performans yönetimine ihtiyaç duyulmaktadır. Performans ölçüm ve kıstas değerlendirmelerine projelerin ilk fazında başlanılması gerekirken, günümüz iş dünyasında sistem canlıya alınırken ya da alındıktan sonra başlanılmaktadır. IT yöneticileri zamandan ödün vermek istememekte ve performans sorununun donanım ekleyerek ya da bazı operasyonel düzenlemelerle aşılacağına inanmaktadırlar. Bu durum ya yüksek kapasite maliyetlerine ya da önemli gecikmelere sebep olmaktadır.

Performans son kullanıcılar için cevap süresi ya da bant genişliği bazında saniyedeki işlem sayısını ifade etmektedir. Şirketler performans hedeflerini gerçekleştirmek için kapasiteye yatırım yapmakta ve sorun durumunda kapasite artırımı çözümüne odaklanmaktadırlar. Bununla birlikte, esas olması gereken performans yönetiminin yazılım döngüsü sürecinde değerlendirilmesidir. Performansın önemli olduğu uygulamalarda sadece geliştirme sürecinde değil, tasarım, test ve bakım aşamalarında da dikkat edilmesi gerekmektedir.

Performans yönetiminde yazılım geliştirme aşamasında teknik performans detaylarının incelenmesi ve kodun performanslı çalışacak şekilde yazılması yoğun yükte çalışan sistemlerin performanslı çalışmalarını sağlamaktadır. Bunun yanı sıra, yük testleri ve stres testleri uygulanarak istatistiksel veriler elde edilip performans değerlendirmeleri yapılabilmektedir.

Bu tezde performans yönetimine odaklanılmış, yoğun yükte çalışan sistemlerde performansı etkileyecek faktörlerin belirlenmesi ve performans testleri uygulamaları üzerinde çalışılmıştır.

Anahtar Kelimeler: yük testi, performans yönetimi, yazılım

ANALYZING BEHAVIOUR DIFFERENCES IN SOFTWARE SYSTEMS ON INTENSIVE LOAD

Presented by: **Melih SAKARYA**

ABSTRACT

IT Systems need to performance measurement which is missing in most implementations to keep managing increasing workloads. Performance Measurement and Criteria Evaluations needs to start right from the first phase of the project but in business, today is that performance is looked at either during preparation of production or once the system goes live. Most IT managers do not wish to compromise on time to market and believe that performance can be managed by adding hardware or some operational arrangement. This situation causes high cost of capacity or significant delays.

Performance expresses response time for end users or throughput in terms of business transactions per second. Enterprises invest in capacity to meet performance targets and and focus on the solution of adding hardware when a performance problem occurs. However, what is truly needed is to treat performance as an integrated aspect of the software lifecyle. On applications which is the performance is a significant issue, the control and evaluation of performance is not only within the development process, is also included in maintenance, design and testing processes.

On performance management, investigation of the technical performance details and writing the code as working on high volume provide performance for the system running at heavy load. Thus, as well as load tests and stress test by applying statistical data acquisition and performance evaluation can be made.

This thesis focus on performance management to identify the factors that influence the performance of systems running at peak load and evaluate performance testing.

Keywords: load test, software, performance measurement, software lifecycle

İÇİNDEKİLER

TABLOLAR LİSTESİ

Sayfa No.

ŞEKİLLER LİSTESİ

1. GİRİŞ

Internet dünyasının gelişmesi ve kullanım oranlarının artması ile birlikte özellikle web ve mobil uygulama katmanlarında performans tarafında bazı sorunlar oluşmaya başlamıştır. Özellikle sistemlerin yoğun ziyaret aldığı dönemlerde uygulamalar yeterli kaynaklara sahip olmadığı zaman, gerek yanıt süreleri gerekse uygulamanın davranışını etkileyebilecek sorunlarla karşılaşabilmektedirler.

Havayolu şirketlerinin erken rezervasyon kampanyaları, belirli kurumlar içerisinde (ÖSYM, Üniversite vb.) yapılan sınav sonuçlarının açıklanması veya bankalar için maaş ödeme günleri, kullanıcı ziyaretlerinin ve işlemlerin yoğun olarak yapıldığın gözlemlendiği dönemlerdir. Sistemlerde olan yavaşlıkların yanı sıra uygulama akışlarında meydana gelen davranışsal değişiklikler ve fonksiyonalitenin tamamlanamaması gibi durumlar ciddi müşteri ve gelir kayıplarına neden olabilmektedir.

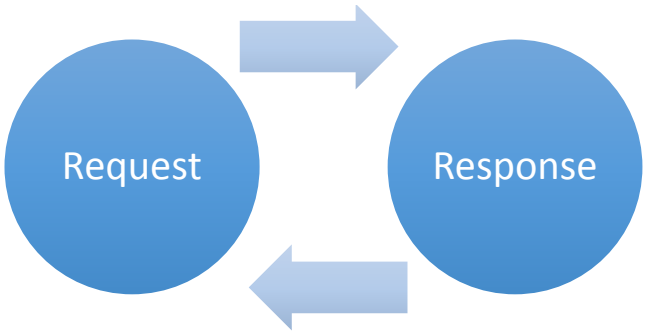
Nah, Fiona Fui-Hoon (2004)' un araştırmasına göre kullanıcıların ortalama sisteme giriş sürelerinin 2 saniyenin altında olması idealdir. Ziyaretçinin ilk karşılandığı ana sayfa gibi ekranlar için 10 saniye üzeri beklemelerde kullanıcıların sisteme girişten vazgeçmelerine neden oldukları gözlenirken iç sayfa gezintilerinde ortalama 2 saniye üzeri beklemelerde kullanıcılar süreçlerini tamamlamadan vazgeçme eğilimi göstermektedirler [1].

Bazı performans metrikleri kullanıcıların davranışlarının yanında hatalı işlemlere de neden olabilmektedir. Bunların başında geliştirme aşamasında hatalı yazılım tasarımı nedeniyle gerçekleşen eş zamanlılık problemleri gelir. Buradaki temel hata aynı veriye erişme veya aynı bellek alanının kullanımı olabilmektedir. Uçak rezervasyonda aynı koltuğun birden fazla kullanıcı ile alınması veya stoktaki ürünlerin stok sayısından fazla satılması bu duruma örnek teşkil etmektedir. Bu tarz problemler ile güncel iş dünyasında karşılaşılabilmekte ve genellikle sistemlerin yoğun olduğu dönemlerde görülmektedirler.

Performans problemlerin sistemin kısmi kullanıldığı zamanlarda sıklıkla karşılaşılmadığından ön görüsü zor olmakta ve aynı hata en çok işlem anında birden fazla yapılmaktadır.

2. İNTERNET YAŞAM DÖNGÜSÜ VE KATMANLAR

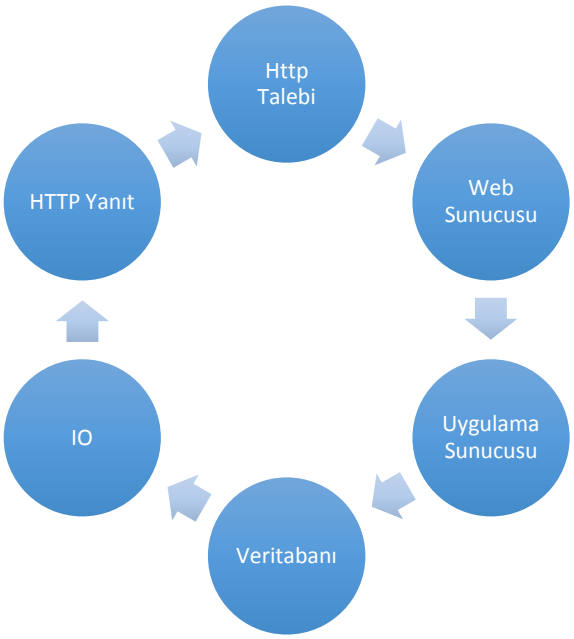
Internet ortamında kullanıcılar tarafından yapılan taleplerin bir yaşam döngüsü bulunmaktadır. Talebin istemci sunucu iletişimi kısmını oluşturan yapı, Request ve Reponse adımlarından oluşmaktadır. Request bir talebe karşılık gelirken Response talep sonrası sunucu tarafından verilecek cevabı ifade etmektedir. Talep döngüsünün tamamı kullanıcının sistemdeki bir çağrısına karşılık gelmektedir.



Şekil.1. Request-response ilişkisi

Request istemci tarafında bir Google Chrome gibi tarayıcı veya mobil ortamda Restful tipinde bir web servis çağırımı ile yapılabilir. HTTP protokolü standartlarında istemci sunucu markası veya tipi ile ilgilenmeden yanıtını beklemektedir. Bu zamanda geçen süre uygulama performansını oluşturmaktadır.

Talebi karşılayan sistem yanıtı üretirken dinamik katmanlar ile haberleşiyor olabilir. Bu noktada mevcut akış içerisinde performans farklılığı yaratacak birçok ürün veya çözümle karşılaşıyor olabilir. Bant genişliği, ağ kapasitesi, web sunucusu, uygulama sunucusu, veri tabanı, disk hızı ve yanıt performansı bu kapsamdadır. Sistem performansı data iletişimindeki bant genişliği ve donanım hızıyla birlikte değişmektedir [2].



Şekil.2. Talep döngüsü

HTTP Talebi: İstemcinin (tarayıcı) ile yaptığı talebin sunucuyu ulaşmasına kadar geçen zamanı ifade etmektedir.

Web Sunucusu: Gelen talebi karşılayan web sunucusunun bu işleme karşılık üreteceği yanıt ile geçen zamanı ifade etmektedir

Uygulama Sunucusu: Dinamik işlem gerektiren bir katmanda yer alınıyor ise (Java veya Net) bu süre içerisinde çalıştırılan uygulamanın harcayacağı zamanı ifade etmektedir.

Veri tabanı: Veri katmanı dinamik olarak işleniyor ise Java gibi uygulamaların veri tabanına bağlanma ve ilgili sorguları çalıştırması ile geçen zamanı ifade etmektedir.

HTTP Yanıt: Gelen talebin sonucunda ilgili verilerle yanıtın üretilmesi ve bu anlamda geçen süreyi ifade etmektedir.

Tüm adımlar için sistemin performansını etkileyecek işlemci, bellek, internet bant genişliği, disk yazma ve okuma hızı, disk doluluk oranı, ağ performansı gibi birçok etmen performans ölçümüne dahil olmaktadır.

3. PERFORMANS ÖLÇÜMLEMESİ VE KULLANILAN METRİKLER

Yazılım dilinde uygulama performansının belirlenmesi için kullanılan kabul görmüş birden fazla tanım bulunmaktadır. Bunlar TPS – Transaction Per Second saniyedeki işlem sayısını ifade etmekte, Response Time cevaplanma süresini ifade etmekte, IOPS - Input/output operations per second giriş çıkış için bir saniyedeki işlem sayısını ifade etmekte ve Throughput belirli zamandaki işlem sayısını ifade etmektedir.

3.1 TPS (Transaction Per Second) – Saniyedeki İşlem Sayısı

Saniyede yapılan işlem sayısını ifade eder. Bu hesaplamada sistemin bir saniye içerisinde yapabildiği işlem hesaplanır. TPS ölçümlemesi yapılırken karşılanacak fonksiyon belirtimi de yapılması gerekmektedir.

- HTTP Anlık Talep/TPS
- Kullanıcı Kaydı/TPS

Her iki karşılaştırma içinde iş tanımları anlık yapılabilecek HTTP protokolündeki talep ve bir fonksiyona karşılık gelen kullanıcı kaydını ifade etmektedir. Benzer olarak veri tabanı kaydı gibi işlemlerde bir TPS ölçütü olarak değerlendirilmektedir. Buradaki planlamada sisteme gelen yük bağımsız olarak hesaplanır. Hesaplanan değerin daha düşük zamanları kapsaması durumunda bu işlemin bir saniye altında tamamlanması beklenir.

3.2 Response Time – Cevaplanma Süresi

Bir talebin cevaplanma süresini ifade eder. Burada sistemin belirli bir talep yükünde cevaplama süresinin ölçümlenmesi hedeflenir. Bu planlama işlem veya talep bazında işlenebilir.

Anlık 1.000 kullanıcı ile birlikte 4 Talep ile kullanıcı kayıt işleminin en az 20 milisaniye ve en çok 6.000 milisaniyede tamamlanması durumunda ortalama tamamlanma süresi 2.000 milisaniyedir. Yapılan kontrol beklenen yoğun yük anında sistemin cevaplama süresinde ne kadar yavaşlamaya neden olduğudur.

Doğru planlama için saniyedeki işlem sayısı ve cevaplanma süresi ölçümleri kullanılabilir olmalıdır. Sistem tasarımcıları ön görülerinden ziyade sistemlerini ölçeklenebilir hale getirmekte ancak olası performans problemleri ölçekleneme yetersiz veya maliyetli olmasına neden olabilmektedirler. Bu durumda yoğun yük altındaki bir sistemde kaynak artırımı yapılsa bile sistem karşılayamaz hale gelebilir veya karşılamak için yüksek maliyetler ödenmek zorunda kalınabilir.

Tablo.1. Talebi Karşılayan Sistem – Süre Karşılaştırması

Karşılayan Sistem	**Geçen Süre**
HTTP Talebi	2 ms
Web Sunucusu	6 ms
Uygulama Sunucusu	10 ms
Veritabanı	30 ms
I/O	2 ms
HTTP Yanıt	2 ms
Toplam Geçen Süre	52 ms

Talep döngüsünde geçen sürelere göre performansı etkileyen noktalar tek bir kullanıcı talebi ile hesaplanırken talep sayısı anlık 500 ve üzerine ulaştığında TPS veya Response Time sürelerinde kartezyen artışlar görülebilmektedir. Bu sebeple TPS rakamını ancak toplam değer 1.000 ms, 1 saniyeye eş gelene kadar anlık kullanıcı sayıları artırılır.

Çıktı raporu örneği sisteme "kullanıcı giriş" işlemi için TPS değeri 80 TPS tir. Bu yöntemin Response Time olarak yorumlanması için anlık kullanıcı sayıları ile birlikte koşum sayıları da verilmektedir. "Kullanıcı giriş" işlemi anlık paralel 500 kullanıcı için 4 tekrar ile başlangıç değeri 52 ms, en yüksek değer 2.200 ms, ortalama geçen süre 900 ms' dir.

4. SİSTEMİN YÜK DURUMLARINA KARŞI TEST EDİLMESİ

Bir sistemde gerçek ortamda yoğun yük altında işlemci, bellek, disk, bant genişliği yetersizlikleri ile karşılaşılabilir. Kullanıcıların aktif kullandıkları bir ortamda bu kaynakları sonradan eklemek önemli sorunlara yol açabileceğinden sistem tasarımcıları benzer ortamlar yaratmak istemektedirler.

Sistemin canlı ortamda devreye alım öncesinde kaynak ihtiyaçlarını öngörebilmek için olası anlık talep beklentilerinin yük testleri ile planlanması gerekir. Yük testlerinin en önemli noktalarından biri olabildiğince gerçek kullanıcı davranışlarına yakın hareketler oluşturulmasını sağlamaktır.

Yük testi planı için aşağıdaki gibi örnek bir senaryo oluşturulmuştur.

- Tek bir sayfa çağırımı
- Anlık 1.000 kullanıcı
- 4 tekrar
- 10 saniye artış süresi

Bu plana uygun olarak sistem anlık 1.000 kullanıcıya 10 saniye her saniye 100' er kullanıcı ekleyerek ulaşacaktır ve her bir kullanıcı için bu işlem 4 defa tekrar edecektir.

Yük ortamlarını oluşturmak için birçok ücretli ve ücretsiz araç bulunmaktadır. Yük ortamı oluşturma araçlarından en çok kullanılanları HP firmasının LoadRunner ürünü, Borland firmasının Silk Performar aracı, IBM firmasının Rational Performance Tester aracı ile Apache grubunun açık kaynak ve ücretsiz JMeter olarak sayılabilir [5] [6].

Tez kapsamında uygulama testleri için açık kaynak ve ücretsiz JMeter aracı kullanılmıştır. JMeter ile aynı zamanda bulut sistemlerde yük ölçeklemesi de yapılabilmektedir.

Sistemde oluşturulan yükün en önemli problemlerinden biri sisteme getirdiği performans probleminin derinlemesine bilinememesidir. Bunun sebebi bu tarz testlerin kapalı kutu olarak tanımlanan sistemin içi ile ilgilenmeyen sadece dışarıdan son kullanıcı benzeri erişimler yapan testler olmasıdır. Yazılım dünyasında

performans problemlerini izlerken problem anında sistem profilini inceleyen uygulama performans izleme araçları kullanılır. Çoğunlukla kullanılan araçlar olarak New Relic, AppDynamics ve Dynatrace verilebilir. Tez içerisindeki örneklerde yük oluşması anında sistem New Relic ile izlenmiş ve raporlar tez içerisinde paylaşılmıştır [8-10].

Uygulama performans izleme araçlarının temel işlevi çalışan uygulamayı bir ajan vasıtası ile izlemek ve performans, veri, hata detayı, veri tabanı erişim detayları gibi bilgileri raporlayabilir ve anlamlandırılabilir hale getirmektir. Bu tarz ajanların uygulama içerisinde %2-%5 gibi bir aralıkta yük getirebildiği görülmektedir.

4.1 Yük Testlerinde Dikkat Edilecek Durumlar

Yük testi yaparken yükün çıktığı makine veya makinelerde kaynak planlaması donanımsal ve sistemsel kaynaklara göre değişkenlik [2] gösterirken sistemin uzakta veya yakında olmasına göre de farklılık yaratmaktadır [3].

Tek bir makineden çıkacak olan yük belirli bir sayıyı aşıyorsa performans problemi yükün çıktığı makineden de kaynaklanabileceği için sonuçlar gerçekçi olmayabilecektir. Örneğin, Intel i5 2.7 GHz işlemcili 4 GB bellek kapasiteli bir makineden 10.000 paralel thread sayısına sahip bir yük çıkmak istenirse gerçeğe yakın sonuçlar alınamayacaktır. Buradaki temel problemler makinede oluşacak paralel thread sayısı, açılacak socket sayısı, bellek kullanımı ve bant genişliğinin yetersiz kalma riski olacaktır. Bu problemlerin yanı sıra yapılacak 100 KB' lık yanıtlar üreten talepler 10.000 kullanıcı için yaklaşık 1 GB anlık veri transferine ihtiyaç duymaktadır. Bu durumda yaklaşık 8 Gbit' lik bant genişliği gerekmektedir.

Sistemin gereken kaynakları karşılayamaması durumunda doğru kullanım için mimarinin dağıtık olarak tasarlanması gerekmektedir. Amazon tarafından sunulan bulut hizmeti olan AWS (Amazon Web Services) platformu sayesinde saat bazında makine kullanımları ile dağıtık bir sistem uygun maliyet ve yüksek kapasite ile kısa zamanda ayağa kaldırılabilir [11].

4.2 Yük Testi Uygulaması

Yük testini uygulamak için istemci olarak kullanılan yük çıkma kapasitesine sahip bir sistem ile yükü karşılayacak bir uygulamaya sahip farklı bir sistem hazırlanmıştır. Her iki platformda konfigürasyonlar aşağıdaki gibi planlanmıştır.

Linux işletim sistemine ait konfigürasyonda herhangi bir ekstra düzenleme yapılmamıştır.

Yük Testi Yapılan Sunucu

- AWS T2 Large Instance
- Linux Ubuntu 14.10
- 64 Bit
- 8 GB Ram
- 2 cCPUs
- 30 GB SSD Disk

Windows işletim sistemine ait konfigürasyonda herhangi bir ekstra düzenleme yapılmamıştır.

Yük Testi Uygulayan İstemci

- AWS T2 Large Instance
- Windows Server 12
- 64 Bit
- 8 GB Ram
- 2 cCPUs
- 30 GB SSD Disk

Yükü karşılayan sunucu sistemi Ubuntu Linux işletim sisteminde PostgreSQL veritabanı, Apache Tomcat Java Servlet sunucusuna sahiptir. Apache Tomcat sunucusu New Relic ajanları ile sarmallanmış ve dinlemeye hazır hale getirilmiştir. Bu durumda anlık olarak olası problemler New Relic web sitesinden izlemeye alınmıştır.

New Relic için Apache Tomcat üzerindeki konfigürasyon aşağıdaki gibi verilmelidir.

New Relic içerisindeki newrelic.yml dosyası içerisinde lisans anahtarı tanımlanmalıdır.

license key [license key]

Apache Tomcat başlangıç tanımları içerisine aşağıdaki gibi tanımlama yapılmalıdır.

-javaagent:/home/msakarya/Desktop/newrelic/newrelic.jar

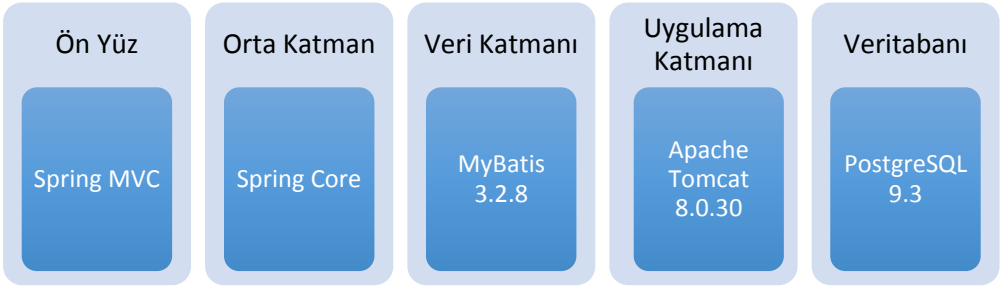
-javaagent:/home/ubuntu/tools/newrelic.jar

İstemci tarafını oluşturan Windows sistem üzerinde Java JDK ile birlikte Apache JMeter 2.13 versiyonu kurulu hale getirilmiştir.

Tüm sistemler Amazon İrlanda bulut merkezinde yer alıyor olup aynı ağ içerisinde yer almaktadırlar. Testler boyunca herhangi bir bant genişliği problemi beklenmemektedir.

Yük testi uygulanacak uygulama mimarisi Java ile tasarlanmış olup DVD Store isimli DVD satış uygulamasıdır. Java tarafında ihtiyaç duyulan bellek kullanım alanı ilk adımda 1.2 GB olarak verilmiştir. Apache Tomcat sunucusu için HTTP talebi karşılama sayısı (HTTP Request Accept Count) 100 ve thread sayısı 200 olarak planlanmıştır.

DVD Satış uygulamasına ait teknik mimari Şekil.4.'de tasarlanmıştır.



Şekil.4. DVD Satış Uygulaması Teknik Mimari

Uygulama tasarımı kurumsal dünyada yoğun kullanımı ve araç desteği nedeni ile Java kullanılarak tasarlanmıştır. Geliştirilen örnekler Java dünyası için tercih edilen teknolojileri içermektedir. Test sürecindeki akışa göre uygulama performansını etkileyecek parametre ve çözümler uygulama içerisine eklenerek performans iyileşmelerinin izlenmesi hedeflenmektedir.

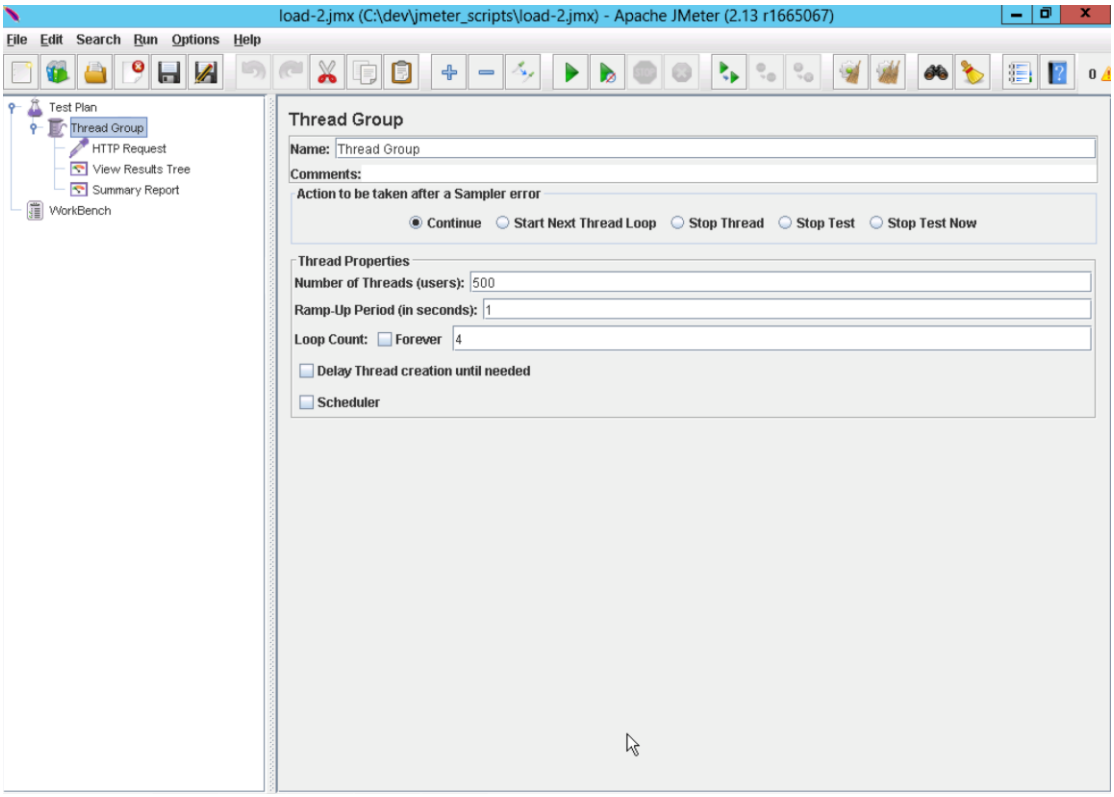
Test planlamasında yük için uygulanacak bilgiler Apache JMeter üzerinde tanımlanıp testlerin koşturulması sağlanmıştır. Talepler HTTP standardında tek bir sayfa çağırımı olarak GET yöntemi ile yapılmıştır. Test planlarında tanımların bazıları aşağıdaki gibidir.

- Anlık Paralel kullanıcı sayısı: Yük anında aynı anda sisteme talepte bulunacak kullanıcı sayısı
- Tekrar Sayısı: Her bir paralel kullanıcının talep edeceği çağırım sayısı
- Toplam Talep Sayısı: Paralel kullanıcıların çağırım sayıları ile birlikte sistemde olacak toplam request sayısı.
- Toplam Talep Sayısı: Sistemde kesilme olmadan oluşturulan talep sayısı
- En Düşük Talep Süresi: Milisaniye cinsinden en kısa sürede yanıt alan talebin toplam süresi.
- En Yüksek Talep Süresi: Milisaniye cinsinden en uzun sürede yanıt alan talebin toplam süresi.
- Ortalama Talep Süresi: Tüm taleplerin milisaniye cinsinden ortalama süresi.
- Hatalı Talep Oranı: Taleplerin karşılaştığı toplam hata oranı.
- Saniyedeki Veri Transferi: Bir saniyede sistem ile haberleşmenin veri karşılığı.

4.2.1 Birinci Seviye Uygulama Testi ve İyileştirme Önerileri

Birinci sevide uygulama test bilgileri aşağıdaki gibidir:

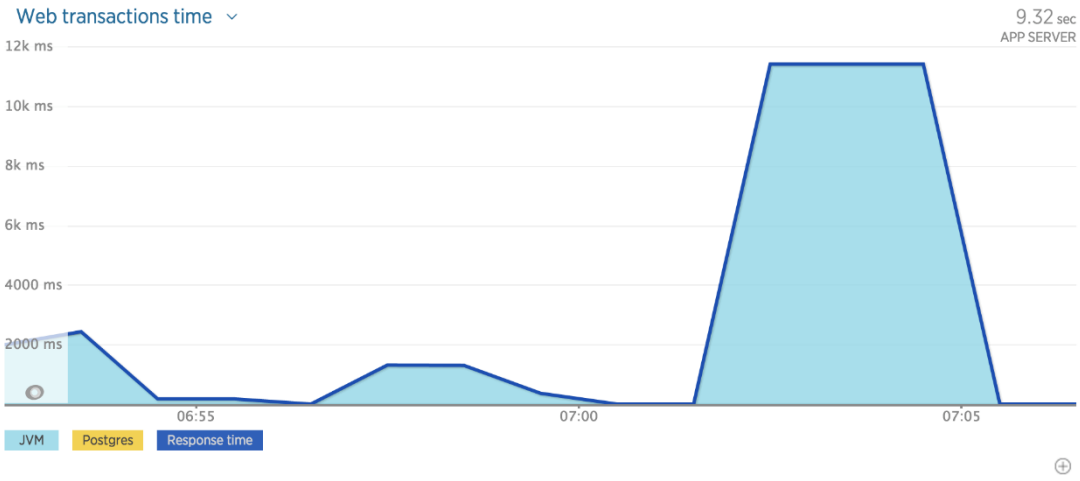
- Anlık Paralel kullanıcı sayısı: 500
- Tekrar Sayısı: 4
- Toplam Talep Sayısı: 2.000



Şekil.3. JMeter Uygulama Testi Ekranı

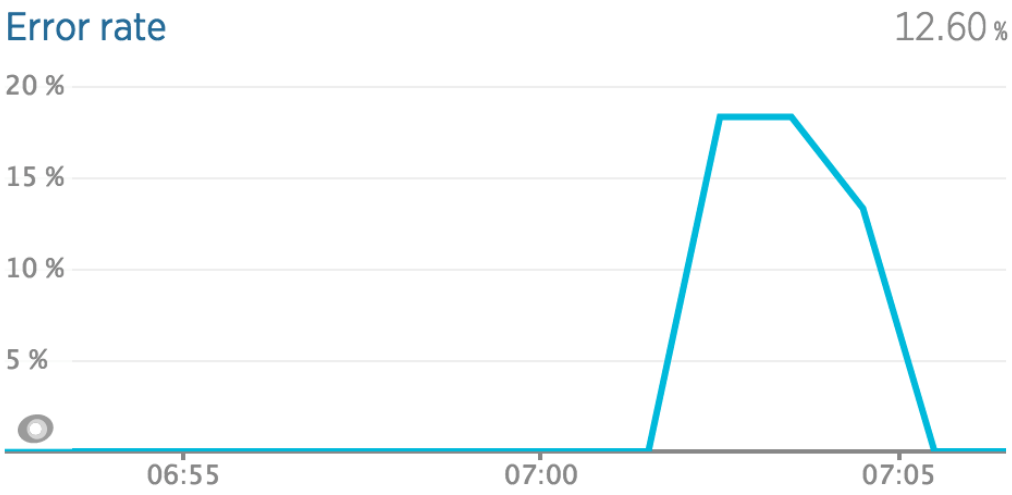
- Toplam Talep Sayısı: 2.000
- En Düşük Talep Süresi: 446 ms
- En Yüksek Talep Süresi: 41.257 ms
- Ortalama Talep Süresi: 13.290 ms
- Hatalı Talep Oranı: %16.10
- Saniyedeki Veri Transferi: 6.518,28 KB

Uygulama performans izleme aracı üzerindeki çıktılar request ve response sürelerinden ayrıştırılmış olarak aşağıdaki tabloda görülmektedir. Bu sonuca göre talebin %95 kadar bir oranı sunucu tarafında beklemede kalmıştır. Bu değer bir web sistemi için kabul edilebilir bir değer olmayıp müşteri memnuniyetine negatif yansıması beklenmelidir.



Şekil.7. Bekleme Süresi Ölçümlendirmesi

Bekleme sürelerinin sunucu tarafında oluşturduğu hatalar Şekil.8.'de yer almaktadır.



Şekil.8. Hata Oranı Ölçümlendirmesi

Uygulama katmanı Java Sanal Makinesi ve Apache Tomcat sunucusu ilk değerlerinde bırakıldığından donanımsal ve sistemsel kaynaklar yeterli olsa dahi yoğun talep anında performans düşüşleri ve hatalar görünmektedir.

İyileştirme fazında sistem kaynaklarının doğru kullanımı için Java Sanal Makinesi bellek kullanımı 4 GB alana çıkarılmışt ır. Apache Tomcat sunucusu için HTTP talebi karşılama sayısı (HTTP Request Accept Count) 250 ve thread sayısı

400 olarak verilmesi uygun görülmüştür. Bu çözümle birlikte sistemde %100 den fazla iyileşme görülmüştür.

Bu değişiklik sonrası değerler:

- Toplam Talep Sayısı: 2.000
- En Düşük Talep Süresi: 115 ms
- En Yüksek Talep Süresi: 32.353 ms
- Ortalama Talep Süresi: 6.603 ms
- Hatalı Talep Oranı: %0
- Saniyedeki Veri Transferi: 13.792,42 KB

Sonuç raporuna göre en düşük talep süresi 446 milisaniyeden 115 milisaniyeye düşmüştür. Bu yaklaşık 4 katına yakın bir iyileşme sağlamış olup sistem karşılamasındaki dar boğazı da rahatlatmıştır.

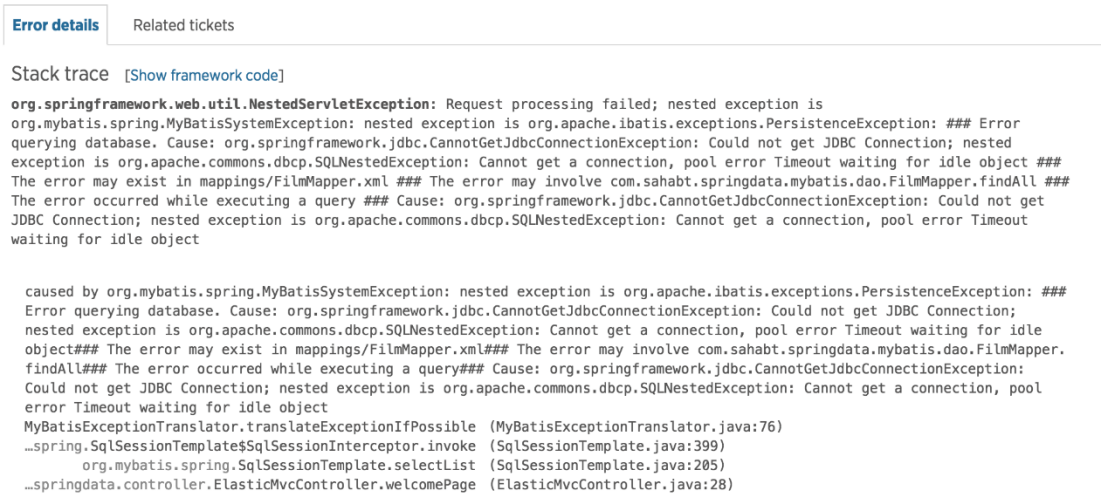
Yoğun yük altındaki sistemlerde sınırsız sayıda thread ile işlem yapması ve bellek alanının yetmemesi gibi durumlarla karşılaşmamak için kabul edebileceği aktif kullanıcı sayılarını kısıtlamıştır. Yapılan yük testinde anlık 500 kullanıcının sisteme girmesi ancak 100' erli olarak kabul edilmekte iken yeni konfigürasyon ile bu sayı 250' ye çıkarılmıştır.

Bununla birlikte bellek yetersizliği nedeni ile Java tarafında uygulama sürekli çöp toplayıcısı ile işlemci kullanımı sağladığından bellek artırımı bu yöndeki darboğaza çözüm olmuştur.

4.2.2 İkinci Seviye Uygulama Testi ve İyileştirme Önerileri

Konfigürasyonel değişikliklerle birlikte uygulama katmanında oluşan iyileşmelere karşın hata oranlarında yükseklikler dikkat çekmektedir. Buna göre sistemde %110 gibi oranda iyileşme yaşanırken hata yüzdesinin yüksekliği sistemdeki kullanıcıların uygulamada geçirdikleri akışı etkileyebilmektedir.

Hata oranı talep yükü ile doğru orantıda artmaktadır. Hata detayına uygulama performans izleme aracı ile bakıldığında Şekil.5.'de detayları görülebilmektedir.



Şekil.5. Hata Detayı Ekranı

Hata detayına göre veri tabanı bağlantı havuzunda sistemin beklediği açık bağlantı bulunamaması nedeniyle bir hata oluşmaktadır. Bunun yanında sistem halen beklenen düzeye yakın bir iyileşme gösterememiş görünmektedir.

Sistemde veri katmanında giden veri tabanı bağlantı havuzu değeri 20 olarak verilmiştir. Buradaki parametreleri artırabilir durumda görülebilirken bunun bir çözüm olmayacağı ve sistemin yoğun kullanımına göre paralelinde bir artış bekleyeceği görülmüştür.

Yazılım dünyasında veri katmanı maliyetli, kararlılık ve performans sorunlarına açık bir nokta olarak görüldüğünden ikinci seviye ön bellekleme yaklaşımı ile bu probleme çözüm önerileri sunulmuştur. Bu yaklaşıma uygun ürünler olarak Hazelcast, Oracle Coherence, MemCached, Ehcache ve Redis gibi çözümler sunulmaktadır [12]. Uygulama içerisinde Redis çözümü ile sorgular bellek içerisine alınmış ve sorgu tekrarlanmasından kaçınılmıştır.

Redis ile temel olarak yapılan ilk sorgu talebi ile birlikte gelen sonuçların bellek üzerinde saklanması olmuştur. Bu sorgu miktarının bellek kullanımını yormayacak miktarda olması beklenmektedir. Eğer ön bellekleme çözümleri aşırı kullanımlara neden olursa bu durumda bellek aşım hataları gibi durumlara neden olabileceklerdir.

Bu çözüm ile sistemde yük durumu engellenmiştir.

- 2.000 olan veritabanı sorgu sayısı minumum değerlere düşürülmüştür.
- Veritabanına yapılan socket bağlantısı ve I/O işlemi azaltılmıştır.
- Bağlantı havuzu meşgul edilmemiştir.
- Socket erişimi minimuma alınmıştır.

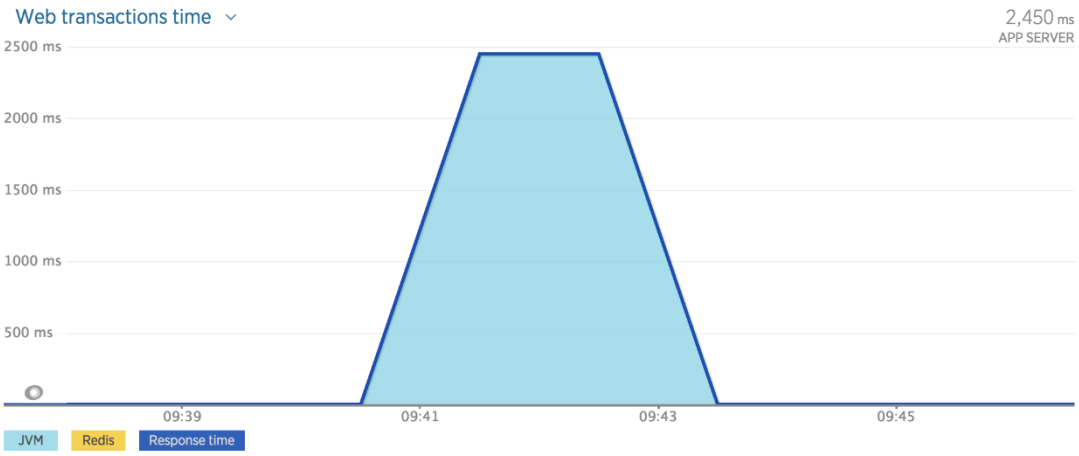
Testler ortam farklılıkları yaşamaması için JMeter ile tekrar çalıştırılmıştır.

- Anlık Paralel kullanıcı sayısı: 500
- Tekrar Sayısı: 4
- Toplam Talep Sayısı: 2.000

Bu değişiklik sonrası çalıştırılan testlerde sonuçlar:.

- Toplam Talep Sayısı: 2.000
- En Düşük Talep Süresi: 10 ms
- En Yüksek Talep Süresi: 10.978 ms
- Ortalama Talep Süresi: 3.317 ms
- Hatalı Talep Oranı: %0
- Saniyedeki Veri Transferi: 31.200,04 KB

Uygulama performans izleme aracındaki rapora göre 2.500 milisaniye altında bir yanıt süresi görülmektedir. Tüm bunların yanında hata oranı %0 olarak verilmektedir.



Şekil.9. Değişiklik Sonrası Bekleme Süresi Ölçümlendirmesi

Kabul edilebilir bir sistemde hata oralarının sıfır olması beklenmektedir. Ancak stres testi gibi sistemi zorlayıcı ve kaynakları son noktasına kadar tüketen bir testte beklenmedik noktaların ne olduğunu izleme adına hatalı durumlar oluşması beklenir. Örneğin anlık en fazla kullanıcı sayısı 800 olarak planlanan bir sistemin 2.000 kullanıcı ile testi yapıldığında veri tabanı veya uygulama katmanında olası hataların ne olacağı gözlemlenmek istenir. Bu durumda kaynakların hangi noktaya kadar ihtiyaçları karşıladığı görülebilir.

4.2.3 Üçüncü Seviye Uygulama Testi ve İyileştirme Önerileri

Yazılım çözümlerinde veri katmanı hareketleri yoğun bir şekilde görülmekte ve veri bütünlüğü için kritik bir durum oluşturmaktadır. Özellikle müşteriye yansıyan veya mali hareketler içeren sistemler veri bütünlüğü veya hizmet kalitesine adına gerektiğinden fazla kaynakla desteklenmek zorunda kalınabilir. Bu durum kurum içerisinde lisans ve kaynak maliyetlerine sebep olabilmektedir.

Büyük veri taşıyan ancak kayıt ve düzenleme gibi veri katmanını değiştirmede çok fazla zaman harcamayan sistemlerde sorgulama işlemleri oldukça zaman harcayabilir durumdadır. Özellikle karmaşık arama yapan sorgular veri kaynaklarında bir süre sonra önemli zorlanmalara neden olabilir. Kargo firmaları için adres veya elektronik ticaret şirketi için ürün arama servisleri bu duruma örneklerdir. Sürekli yeni ürün eklenmemesine karşın ürün sorgulamaları gün içerisinde milyonlarca defa yapılabilmektedir.

Arama işlemlerinde klasik veri tabanlarındaki ilişkisel mimari ve indeksleme yapısı ACID prensipleri de düşünüldüğünde performans sıkıntılarına neden olmaktadır. Bu duruma çözüm olarak yeni nesil arama motoru yaklaşımları getirilmiştir. Açık kaynak Apache Solr ve ElasticSearch en çok kullanılan ürünlerdir. Bu ürünlerin temel özelliği ACID standartlarını desteklemek yerine verinin doğru indekslenmesi ve sorgulanmasını hedef almasıdır. Bu sebeple verinin güvenliği veya bütünlüğü ile ilgilenilmemektedir.

Sistemde örnek bir arama çağırımı yapıldığında aşağıdaki gibi bir sorgu ekranı getirmektedir. Çağırım direkt olarak veri tabanına JDBC seviyesinde aşağıdaki sorgu ile gitmektedir.

select * from film where lower(title) like ‘A*’

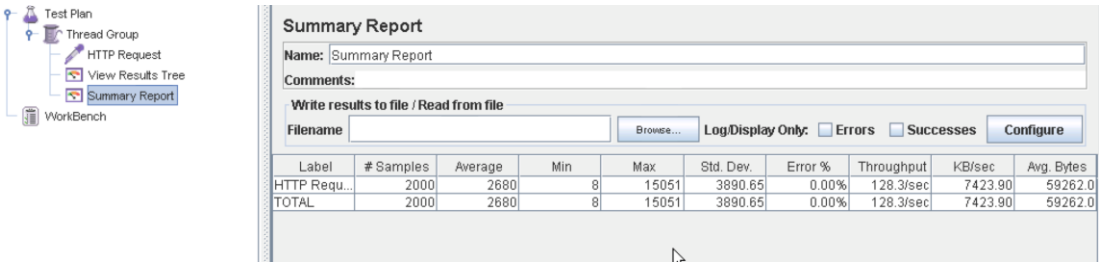
Bu çağırım her seferinde ön bellekleme imkanı bulamadan veri tabanına gönderilecektir. Bunun sebebi veri katmanındaki büyüklüğe göre ön belleklemenin verinin tamamını karşılayamaması veya karşılasa bile veri tabanı arama fonksiyonlarının yeterince esnek olmayacağıdır.

Testler ortam farklılıkları yaşamaması için aşağıdaki şekilde tekrar çalıştırılmıştır.

- Anlık Paralel kullanıcı sayısı: 500
- Tekrar Sayısı: 4
- Toplam Talep Sayısı: 2.000

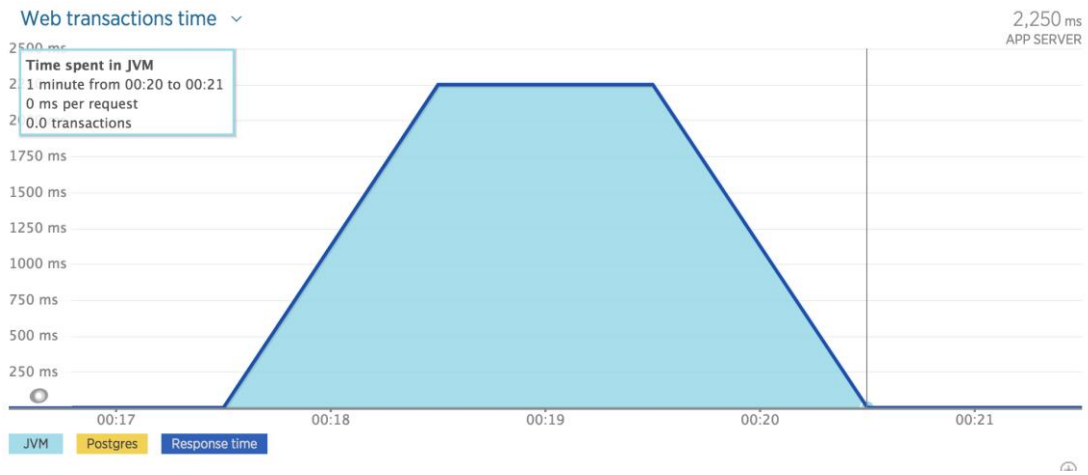
Arama işlemleri veri tabanı üzerinden yapıldığında sonuçlar Resim.2.’de verilmektedir.

- Toplam Talep Sayısı: 2.000
- En Düşük Talep Süresi: 8 ms
- En Yüksek Talep Süresi: 15.051 ms
- Ortalama Talep Süresi: 2.680 ms
- Hatalı Talep Oranı: %0
- Saniyedeki Veri Transferi: 7.423,57 KB



Resim.2. Peformans Testi Özet Raporu

Uygulama performans aracına göre sonuç raporu da Şekil.10.'da görünmektedir. Buna göre talep yanıtlama süreleri sunucu seviyesinde 2.500 milisaniyenin üzerinde çıkmıştır.

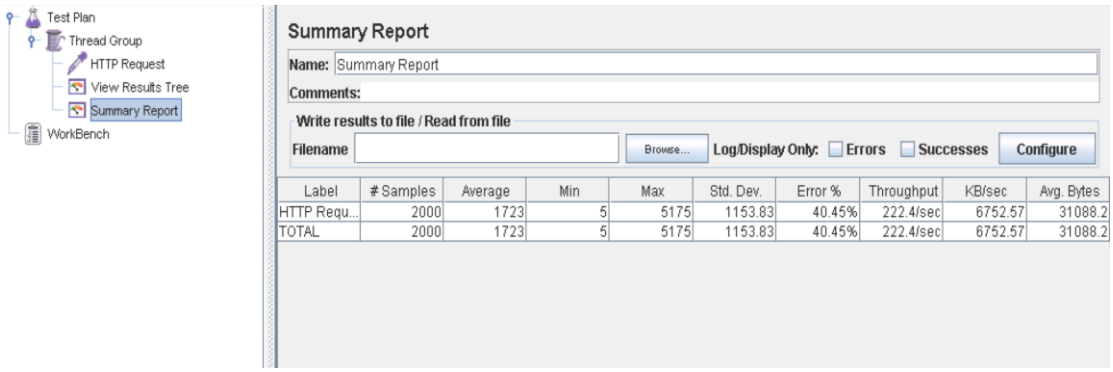


Şekil.10. Uygulama Performansı Bekleme Süresi Ölçümlendirmesi

Sorgular bir sonraki aşamada veri tabanından ayrıştırmak için ElasticSearch üzerine alınmıştır. Bu durumda ilk indekleme işlemi sonrasında sorguların veri tabanına gitmemesi beklenmektedir. Performans problemine yol açabilecek nokta ise veri tabanındaki transaction sayısının yüksek olması durumunda ElasticSearch indekslerinin sürekli güncellenmesi olabilir. Bu durumda entegrasyonların sorgu sayısı yüksek ve karmaşık arama algoritmaları içeren bölgelere yapılması beklenmektedir.

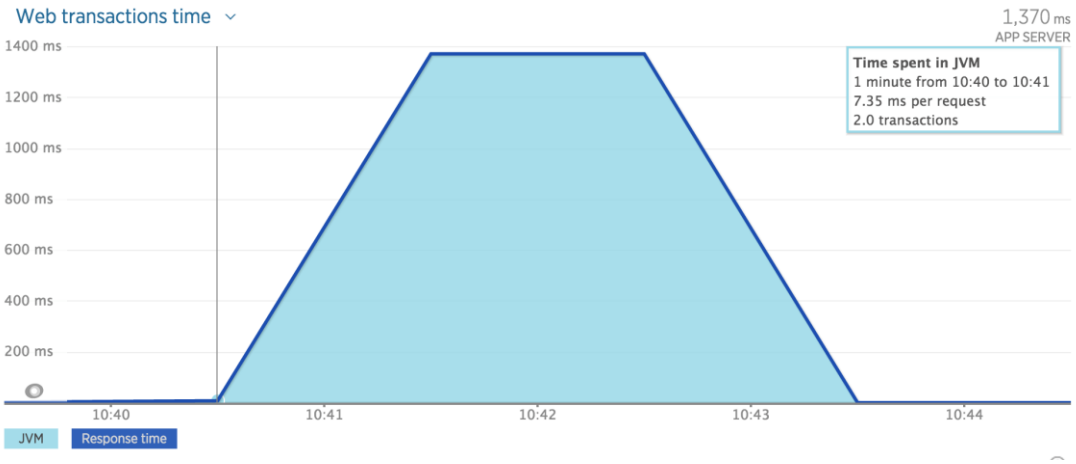
Bu değişiklik sonrası çalıştırılan testlerde sonuçlar Şekil.6.'daki gibi oluşmuştur.

- Toplam Talep Sayısı: 2.000
- En Düşük Talep Süresi: 5 ms
- En Yüksek Talep Süresi: 5.175 ms
- Ortalama Talep Süresi: 1.723 ms
- Hatalı Talep Oranı: %0
- Saniyedeki Veri Transferi: 6.752,57 KB



Şekil.6. Değişiklik Sonrası Peformans Testi Özet Raporu

Uygulama performans aracına göre sonuç raporu da aşağıdaki gibi görünmektedir. Buna göre talep yanıtlama süreleri sunucu seviyesinde 1.400 milisaniyenin altına indirgenmiştir.



Şekil.11. Talep Süresi Ölçümlendirmesi

5. EŞ ZAMANLILIK PROBLEMLERİ

Yazılım sistemlerinde yoğun yük altında iken performans sorunlarının yanında fonksiyonel hatalarla da karşılaşabilme riski bulunmaktadır. Bunun genel olarak yazılım tasarımından kaynaklandığı görülmektedir. Örneklendirmek gerekirse bir havayolu rezervasyon sisteminde aynı koltuğun iki kere satılması veya kredi hesaplama anında hatalı sayısal değerlerin işleme alınması gibi düşünülebilir.

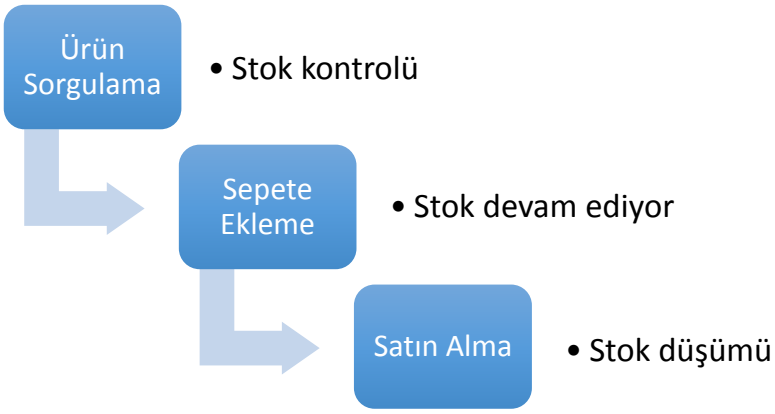
Bu tarz eş zamanlılık problemleri veri katmanına erişimden kaynaklanabileceği gibi aynı bellek nesnesi içerisindeki kod bloklarının birden fazla işlem birimi ile işletilmesinden kaynaklanabilir. Bu tarz problemlerin nadiren hata programsal hatalar üretmesi nedeniyle fark edilmesi oldukça güç olabilmektedir.

5.1 Veri Katmanına Eş Zamanlı Erişim

Uygulama veri katmanında tutulan bilgiler tekil erişime sahip olmak zorunda olabilir. Bunlar ürüne ait stok bilgisi, banka hesabındaki para miktarı veya uçak içerisinde rezerve edilebilecek koltuk olabilir. Sistem stokta olandan fazla ürün satar veya iki ayrı yolcuya aynı koltuğu verirse bu büyük ihtimalle bir hata üretmeyecek ve ürünün tedarik sürecinde veya uçuş anında problem görülecektir. Bu tarz problemlerin geri dönüşü ve müşteriyi ikna etme süreci oldukça problemli olmaktadır.

Bu tarz problemler genel itibari ile veri katmanındaki stok veya durum kontrolü sonrasında ilgili veriye farklı kaynaklardan erişime izin verilmesinden veya değişim kontrollerinin yapılmamasından dolayı kaynaklanmaktadır.

Probleme ait benzer bir süreç Şekil.12. deki şemada yer almaktadır.

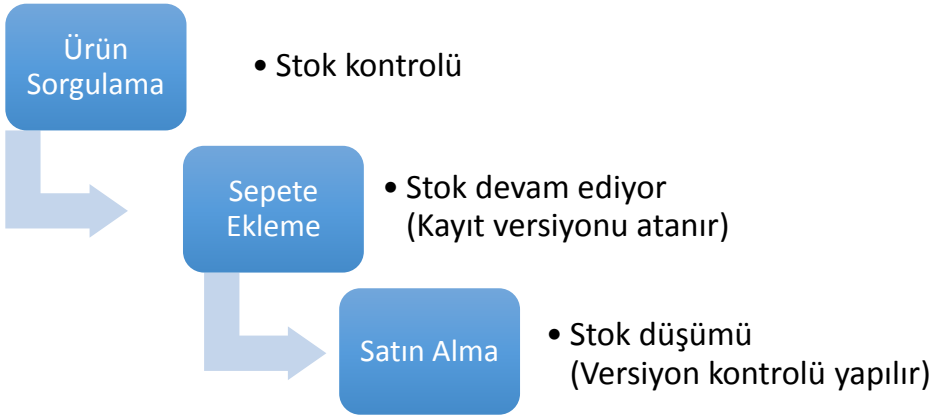


Şekil.12. Kilitsiz Akış Şeması

Şemadakine bezer durumlarda sistem stok düşümü öncesi veya sonrasında herhangi bir durum kontrolü yapılmamaktadır. Bu duruma karşın iki tip kilit mekanizması kullanılabilir. Bunlar iyimser kilitleme (Optimistic Locking) ve kötümser kilitleme (Pessimistic Locking) kilit mekanizmaları olarak adlandırılır.

5.1.1 İyimser Kilitleme (Optimistic Locking)

Veri katmanında uzun süreli bir kilitleme sağlamaz. Verinin değişimi kaydı veya değişimi anında versiyon durum kontrolü yapılır. Bu anlamda kötümser kilitleme mekanizmalarına göre daha performanslı olmasına karşın durum bildirimini son anda kullanıcıya bildirir. Buna ilişkin adımlar Şekil.13. teki şemada yer almaktadır.



Şekil.13. İyimser Kilitleme Akış Şeması

Bu durumda satın alma gerçekleşmesi anında eğer kayıtta herhangi bir değişim olduysa kullanıcıya ürün stokta kalmadı gibi bir uyarı vermesi beklenmektedir.

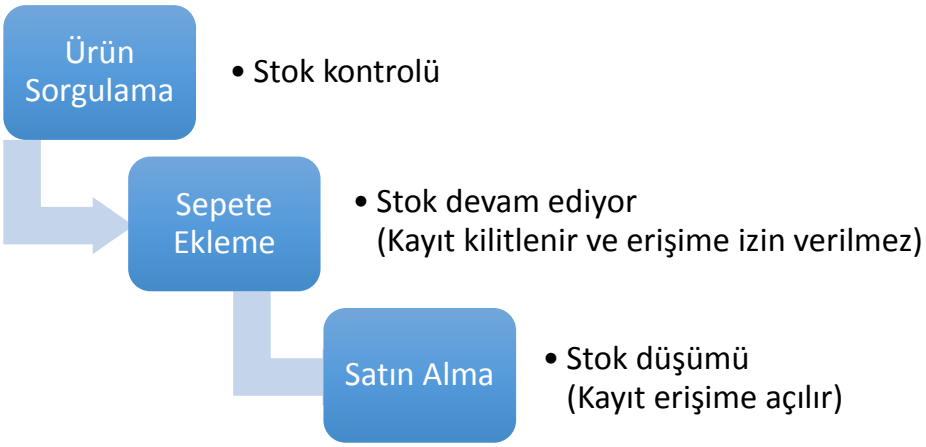
İyimser kilitleme uygulamalarına örnek olarak Java dünyasında veri katmanı erişiminde kullanılan JPA (Java Persistence API – Java Kalıcılık Kütüphanesi) içerisinde yer alan @Versiyon tanımı örnek verilebilir [13]. @Versiyon tanımı yapılan Java varlık nesnesi (entity) sayısal kayıt her bir veri tabanı değişim işleminde (transaction) yeni bir versiyon üretir. 0 ile başlayan bu versiyon her bir adımda artırılarak kendisini yeniler. Veri katmanındaki değişim anında JPA katmanı bu değişimleri kontrol eder ve eşleştiremediği durumlarda kullanıcıya OptimisticLockException sınıfında bir hata fırlatır.

Bu işlem her veri tabanı işlemine karşılık yeni bir işlem daha yarattığından yoğun sistemlerde minumum düzeyde de olsa bir performans problemine neden olabilir.

5.1.2 Kötümser Kilitleme (Pessimistic Locking)

Veri katmanındaki kayıt için belirli süre kilitleme sağlar. Bu sayede aynı veriye ikinci bir erişim veya değişime izin vermez. Bu sistem daha güvenilir olmasına karşın performans problemlerine neden olabilmektedir.

Kilit işlemi okuma veya yazma seviyelerinde yapılabilmektedir. Bu duruma göre kullanıcı veriye erişim için bekleyebilir veya değişime izin verilmeyebilir. Bu durum Şekil.14. teki şemada yer almaktadır.



Şekil.14. Kötümser Kilitleme Akış Şeması

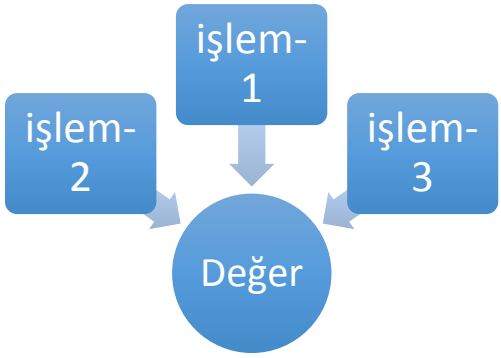
Kötümser kilit mekanizmalarında ikinci kullanıcılar sisteme erişememesi ve performans problemlerinden dolayı belirli bir zaman aşımı süresi verilmesi tavsiye edilmektedir.

Yine Java tarafındaki JPA kütüphanesinde kilit mekanizmaları için gerçekleştirimler yer almaktadır [14].

5.2 Kod Bloklarına Eş Zamanlı Erişim

Bellek alanında aynı nesneye ait kod bloklarına erişimler mümkündür. Örneğin hesaplama yapan bir uygulama metodu veya nesne değerine aynı anda iki

işlemin erişmesi ile hesaplama beklenenden farklı sonuçlar verebilir. Bu durumun bir simülasyon ile yaratılması oldukça güç olabilmektedir.



Şekil.15. Kötümser Kilitleme Akış Şeması

Şekil.15. te yer alan şemada üç ayrı işlem tek bir nesneye erişmektedir. Bu tarz nesneler aynı bellek alanını göstermekte veya sabit alanlı nesne tanımları olabilmektedir. Bu durum Java dilinde static anahtar kelimesi ile yaratılabilir [15].

```
public class Hesap {

    static double para = 100;

    public void paraCek(double miktar) {
        para -= miktar;
    }

    public void paraYatir(double miktar) {
        para += miktar;
    }
```

Şekil.16. Sabit Alanlı Nesne Erişimi

Şekil.16. da yer alan tanımda para değeri static anahtar kelimesi sayesinde bellekte sabit hale getirilmiştir. Bu durum Hesap gibi bir sınıfta programsal olarak bir hata yaratmasa da fonksiyonel olarak problemlere neden olacaktır.

```
public static void main(String[] args) {
    Hesap hesap1 = new Hesap();
    Hesap hesap2 = new Hesap();

    hesap1.paraYatir(100.0);
    hesap1.paraYatir(200.0);

    hesap2.paraCek(100);

    System.out.println("hesap1.para : " + hesap1.para);
    System.out.println("hesap2.para : " + hesap1.para);
}
```

Şekil.17. Nesne Davranış İşlemi

Şekil.17. de yer alan programın çıktısı aşağıdaki gibi görülmektedir.

hesap1.para : 300.0

hesap2.para : 300.0

Bu sonuca göre hesap1 ve hesap2 nesnelerindeki hesap durumu 300 olarak görünmektedir. hesap2 nesnesinde paraYatir metodu hiç çalıştırılmamasına karşın para artışı görülmekte ve tutarsızlık yaratmaktadır. Bunun nedeni hesap1 ve hesap2 nesneleri için para değerlerinin aynı referans alanlarını göstermesidir. Dolayısı ile para nesnesine static tanım vermek programsal olarak bir hataya neden olmamasına karşın tasarımsal olarak problem yaratmaktadır.

Kod bloklarına eş zamanlı olarak erişimde bir benzer problem nesne erişimden farklı olarak aynı kod bloklarının işlenmesidir. Aşağıda yer alan Şekil.18. deki örneğe göre static tanımı verilmiş paraCek metodu bellek üzerinde tek bir nesne üzerinde adreslenmiştir. Bu durumda birden fazla işlem talebine aynı anda ve aynı alanda işletilmek üzere hatalı çıktılar verebilme riski bulunmaktadır.

```
public static void paraCek(double miktar) {
    if(para >= miktar){
        para -= miktar;
    }
    else{
        System.out.println("Yetersiz bakiyeniz bulunmakdatır");
    }

}
```

Şekil.17. Blok İşletimleri

Bu durumda sistem programsal bir hata vermemesine karşın fonksiyonel anlamda hatalı sonuçlar verebilme riski ile karşı karşıyadır. Bu tarz tasarımsal hatalar çok nadir tekrarlandığından tespit edilmesi de oldukça zordur.

Problemin ana noktasını veri değişimleri oluşturmaktadır. Verinin okunması sabit nesne alanları için bir problem teşkil etmemektedir. Ancak veri değişimleri sonrasında yapılan işlemlerde sabit alanlı metodları eş zamanlı erişimlere kapatmak en doğru yöntem olarak görülmektedir.

Java programlama dilinde bu tarz eş zamanlı erişimleri önlemek için farklı yöntemler bulunmaktadır. Bunlar synchronized anahtar kelimesi veya Java Concurrency API olabilir [16].

Şekil.18. de yer alan problemin çözümü için aşağıda yer alan Şekil.19. daki gibi synchronized anahtar kelimesi ile metod eş zamanlı erişime kapatılabilir.

```
public synchronized static void paraCek(double miktar) {
    if(para >= miktar){
        para -= miktar;
    }
    else{
        System.out.println("Yetersiz bakiyeniz bulunmakdatır");
    }

}
```

Şekil.19. Eş Zamanlı Erişime Kapama

Ancak bu tarz çözümlerde tüm metodun erişime kapatılması problem yaratıyorsa benzer bir şekilde synchronized bloğu oluşturulabilir.

```
static double para = 100;

static Object kilit = new Object();

public static void paraCek(double miktar) {
    synchronized (kilit) {
        if(para >= miktar){
            para -= miktar;
        }
        else{
            System.out.println("Yetersiz bakiyeniz bulunmakdatır");
        }
    }

}
```

Şekil.20. Kilit Bloklar

Uygulanan bu yönlemlerin yanında Java 1.5 versiyonu ile gelen Java Concurrency API olduça detaylı özellikler sunar [16]. Sistem tasarımına ve ihtiyaçlara göre örneklendirmek çok daha doğru olacaktır.

5. SONUÇLAR VE ÖNERİLER

İnternet kullanımının sürekli artması ve sistemlerin karmaşıklaşması ile birlikte performans problemleri önemli sorunlar teşkil etmektedir. Her ne kadar donanımsal anlamda verimlilik artsa da kullanıcı sayısına paralel olarak kaynakların artırılması lisans ve donanımsal satın alma maliyetleri yaratmaktadır.

Yazılım mimarileri tasarlarken benzer teknoloji seçimlerinin farklı performans sonuçları verdiği görülmektedir. Tüm bunların yanında yoğun yük içeren sistemler için tasarlanmış yeni nesil araçların kullanımı göz ardı edilmektedir.

Tez içerisinde geçen performans uygulamalarında aynı donanım ve yazılım kaynakları ile yaklaşık olarak 10 katı performans iyileşmesi görülmüştür. Yapılan üç iyileşme fazında ilk olarak uygulama katmanı doğru bellek ve işlemci konfigürasyonuna alınmıştır. İkinci fazda veri kaynağına giden katman ön bellekleme ile minumum seviyeye çekilmiştir. Son aşamada ise arama ve sorgu gibi işlemler standart veri tabanı mimarisinden yeni nesil veri indeksleme mekanizmalarına alınmıştır. Tüm bunların yanında performans nedenli hatalar %14 seviyesinden %0 oranına alınmıştır.

Tezin diğer bölümünde eş zamanlı erişilebilirlik problemlerinden bahsedilmiş ve çözüm önerileri sunulmuştur. Buna göre sistem tasarımlarında yer alan hataların üzerinde durulmuş problemlerin nedenlerine değinilmiştir.

Tez boyunca elde edilen deneyimler ve bilgiler ışığında yoğun yük içeren sistemlerin kaynak artırımından ziyade doğru teknik mimari ve teknoloji seçimleri ile çok daha performanslı, sorunsuz ve maliyet avantajlı hale getirilebileceği gözlenmiştir.

Bu sayede kullanıcılara daha hızlı ve kaliteli hizmet sunabilecek müşteri kaybı engellenecek hale gelmesi beklenmektedir.

KAYNAKLAR

[1] Nah, F.: " A study on tolerable waiting time: how long are Web users willing to wait? Behaviour & Information Technology, forthcoming", (2004).

[2] Hoxmeier, J.A. and DiCesare, C.,,"System response time and user satisfaction: an experimental study of browser-based applications." (2000).

[3] Apache JMeter Best Practices (2015)

http://jmeter.apache.org/usermanual/best-practices.html

[4] Kiran, Sandhya, Akshyansu Mohapatra, and Rajashekara Swamy. "Experiences in performance testing of web applications with Unified Authentication platform using Jmeter." *Technology Management and Emerging Technologies (ISTMET), 2015 International Symposium on*. IEEE, 2015.

[5] Open Source and Commercial Performance Testing Tools (2015)

[6] Open Source Load Testing Tools Comparison: Which One Should You Use ? (2015)

[7] Performance Testing Tool Comparison (2015)

http://www.shooter-smith.co.uk/performance-testing-tool-comparison/

[8] Sari, Rheysa Permata. "Integration of Key Performance Indicator into the Corporate Strategic Planning: Case Study at PT. Inti Luhur Fuja Abadi, Pasuruan, East Java, Indonesia." *Agriculture and Agricultural Science Procedia*3 (2015): 121-126.

[9] Application Performance Management (APM) and Monitoring Tool Comparison (2015)

https://www.extrahop.com/apm-vendor-comparison/

[10] 20 Top Server Monitoring & Application Performance Monitoring (APM) Solutions (2015)

http://haydenjames.io/20-top-server-monitoring-application-performance-monitoring-apm-solutions/

[11] Amazon EC2 Pricing (2015)

https://aws.amazon.com/ec2/pricing/

[12] Top NoSQL Key Value Cache Databases (2015)

http://www.bigdataanalyticstoday.com/top-nosql-key-value-cache-databases/

[13] JPA Version Annotation (2015)

https://docs.oracle.com/javaee/6/api/javax/persistence/Version.html

[14] The Java EE 6 Tutorial - Lock Modes (2015)

[15] Understanding Class Members (2015)

https://docs.oracle.com/javase/tutorial/java/javaOO/classvars.html

[16] Java Concurrency API

EKLER

- EK-1 DVD Satış Java Projesi
- EK-2 DVD Satış Java Projesi
- EK-3 DVD Satış Java Projesi
- EK-4 JMeter Ürün Listesi Projesi
- EK-5 JMeter Ürün Sorgu Projesi
- EK-6 Eş Zamanlı Erişilebilirlik Uygulamaları Java Projesi

ÖZGEÇMİŞ

2002 yılından beri sektörde Java ve orta katman konularında yazılım geliştirici, eğitmen ve danışman olarak çalışmaktadır. Şu anda Saha Bilgi Teknolojileri bünyesinde yazılım test otomasyon süreçleri ile yazılım geliştirme yaşam döngüsü, Java mimari danışmanlığı ve farklı ürün aileleri konularında danışman ve girişimci olarak iş hayatına devam etmektedir. Halen Beykent Üniversitesi Fen Bilimleri Enstitüsünde yüksek lisansına devam etmektedir.

Melih SAKARYA