Sunucu İstek Yoğunluğunun Multithread İle Kontrolü

Kocaeli Üniversitesi Bilgisayar Mühendisliği Yazılım Laboratuvarı 1- 2. Proje

Mert Bilgiç
180202116
bilgic.mert44@gmail.com

Özet

Sunucu İstek Yoğunluğunun Multithread İle Kontrolü Projesi MainServer'e gelen istekleri alt serverlere dağıtarak yük dengelemektedir.
Bu işlem yapılırken alt sunuculardaki yük oranı %70 i aşarsa alt sunucu bölünür ve içindeki istekler yeni sunucu ile paylaştırılır.
Eğer başlangıçtaki sunucular dışında herhangi bir sunucunun yoğunluğu %0 a düşerse bu sunucunun çalışan threadleri durdulur ve sunucu silinir.
Sunucuların kapasitesi canlı olarak ekrana basılır.Kapasiteler thread yardımıyla sürekli olarak takip edilir

I. Giriş

Sunucu İstek Yoğunluğunun Multithread İle Kontrolü Projesi 6 sınıfdan oluşmaktadır.

- Main Class default olan sunucuların oluşturulduğu ve threadlerin başlatıldı class
- Server Class Main ve Sub serverlar için ortak olan özellikleri ver fonksiyonları barındıyor
- MainServer Class Server classını extends ediyor.Üç ana fonksiyona sahiptir.
- SubServer Class Server classını extends ediyor.İki ana fonksiyona sahiptir.
- ThreadManager Class threadlerin oluşturduğu ve fonksiyonlarına sahiptir.
- RequestData Class listemiz için verileri tutarken bu sınıftan türetiyourz.

II. Temel Bilgiler

Program JAVA programlama dilinde geliştirilmiş olup geliştirme ortamı olarak Apache NetBeans IDE 11.1 kullanılmıştır.

III. Tasarım

Sunucu İstek Yoğunluğunun Multithread İle Kontrolü Projesinin geliştirilme aşamaları altta belirtilen başlıklar altında açıklanmıştır.

A. Algoritma

Sunucu İstek Yoğunluğunun Multithread İle Kontrolü Projesinde işletim sistemleri dersinde gördüğümüz produce consume probleminin çözümünden esinlenerek yapılmıştır.

B. Yöntem

Sunucu İstek Yoğunluğunun Multithread İle Kontrolü Projesi'ni OOP prensibine uygun şekilde yazmaya çalıştık bunun için 6 farklı sınıf oluşturduk.Ortak özelliklere sahip sınıflar için kalıtımı kullandık.

Program çalıştırıldığında bir MainServer iki SubServer default olarak oluşturulur.İlerleyen satırlarda ThreadManager sınıfında bir obje oluşturulur ve toplamda main thread dahil 9 thread çalıştırılır.

Bu serverin kendi response request işlemleri için bekleme süreleri oluşturulan objelerden alınır.Threadler belirlenen zaman aralıklarıyla kendileri ile ilgili fonkisyonları çalıştırarak sunucuların durumlarını kontrol eder.Threadlarin eşzamanlı çalışması sırasında oluşan problemleri engellemek için synchronized yönteminde yararlanılmıştır.Bu sayede requestler eklerken veya response dönerken oluşabilecek problemlerin önüne geçilmiş oldu.

C. Pseudocode

1.Default sunucuları oluştur. ThreadManager sınıfından obje oluştur.

2. tm.startMainRequest(); fonksiyonunu çağrılıyor ve thread başlıyor random sayı oluşturuyor. requestCount =random.nextInt (getMaxRequestCount()); random oluşan request sayınsın sunucunun kapasitesini aşıp aşmadığını kontrol et if (getRequestData().size() + requestCount <= getCapacity()) içerdeki toplam request sayını güncelle setTotalRequest(requestData.size());

3. tm.startServerBalance() fonksiyonunu çağrılıyor ve thread başlıyor ilk önce MainServer'da dağıtım için yeterli request var mı kontrol ediyor yok ise Thread.sleep(100); Thread bekletiliyor. Devamında random olarak serverInxdex belirleniyor. Oluşturduğumuz listenin içindeki subserverlardan biri seçiliyor.Random olarak belilenecek request sayısı belirleniyor.Belirlenen request sayısı requestCountControl(int size, int requestCount) fonksiyonunda geçiyor.Sunucunun durumuna uygun request sayısı belirnelerek random olarak seçilmiş sunucuya ekleniyor.Thread'in sürekli olarak devam etmesini engellemek için ThreadManager.balance.setPriority(Thread.NORM _PRIORITY); kesilip kesilip kemiyeceğini ayarlıyoruz.

4. tm.startMainResponse(); fonksiyonu çağrılıyor ve thread başlıyor.Main serverin içindeki response fonksiyonu request datasındaki verileri boşaltarak response yapmış oluyor. Thread'in sürekli olarak devam etmesini engellemek için ThreadManager.balance.setPriority(Thread.NORM _PRIORITY); kesilip kesilip kemiyeceğini ayarlıyoruz.

5. tm.startDefaultResponse();fonksiyonu çağrılıyor ve thread başlıyor. Alt sunucuların response işlemleri burada objeleri kullnarak yapılıyor.Alt sunucuların %70 dolduğunda alt sunucu oluşcağında bu işlemi bir fonkisyona yaptırdırdık. public void create(Server server) {t2 = new Thread(new Runnable() {@Override public void run() {server.response();}});t2.start(); threadResponse.add(t2); şeklide threadResponse listemize ekleniyor bu sayede ilerde kapatacağımız threadleri bir listede tutuyoruz.

6. tm.startListThread();fonksiyonu çağrılıyor ve thread başlıyor.Bu thread ile sunucularımızın ne durumda olduğunu ekrana basıyoruz.Main içinde oluştuduğumuz static liste ile tek tek dolaşıp ekrana basıyoruz.
for (int i=0;i<Main.server.size();i++) {server=Main.server.get(i);System.out.println(serve r.getServerName() + " "server.getTotalRequest() + " %"+percent(server.getTotalRequest(),server. getCapacity()));}

7. tm.capatityControlThread();fonksiyonu çağrılıyor ve thread başlıyor.Sistemdeki çoğu işimizi bu thread yardımı ile yapıyoruz.Sunucuları for döngüsü ile dönerek tek tek durumları kontrol ediliyor.Bu işlem sırasında ana sunucu kontrol edilmiyor.Eğer %70'i dolan sunucuya denk gelirsek divideServer(server); yardımıyla bölme işlemini başlatıyoruz .int startIndex = server.getRequestData().size() / 2;int finishIndex = server.getRequestData().size(); ile sunucunun bölüneceği noktalar belirleyior ve silinyor. createNewServer(temp); fonkiyonu çağrılıyor. int size = Main.server.size();Main.server.add(new SubServer("SubServer-" + Stsize, 5000, 300, 2500, (size + 1), false, 150, 0)); Main.server.get(size).requestData.addAll(requestDa ta);create(Main.server.get(size));nameSize++; kullanılan değişken yardımıyla yeni sunucumuz oluşturulup sunucu listemize ekleniyor. server.response(); gönderilen ındex'e göre obje oluşturlup response işlemi oluşturuluyor. threadResponse.add(t2); ile ilerde sunucunun kapatılması gerektiğinde stop işlemini bu liste yardımıyla yapıyoruz.

8. tm.closeControlThread();fonksiyonu çağrılıyor ve thread başlıyor. for (int k = 3; k < subServer.size(); k++) {server = subServer.get(k); double percent = (server.getCapacity() * 0.7); double currentPer = (server.getRequestData().size() * 0.7);Sunucu kapatma işlemini gerçekleştiricekif (currentPer == 0) { stopResponseThread(k) close fonksiyonu default olan sunucularımızı kontrol etmiyor.Bunun nedeni bu sunucular hiç kapatılmayacak.Sıra sunucuların durumları kontrol ediliyor. threadResponse.get(serverIndex).stop(); threadResponse.remove(serverIndex);Main.server.g et(serverIndex).responseData.size());Main.server.re move(serverIndex); işlemleri sunucumuzdan siliyoruz.stop işlemini gerçekleştiriyoruz

IV. Sonuçlar

Uygulama çalıştırıldığında 1 adet ana sunuc ve 2 adet alt sunucu bulunmaktadır.Bu sunucuların threadleri başlatılarak belirtilen aralıkla response request işlemleri başlatılıyor.Kapasiteye göre alt sunucu oluşturulup kapatılıyor.Sunucuların durumları listeleniyor.

V. Ekran Görüntüsü

```
run:
///////////////// Main Thread Başladı
///////////////// Start Main Request Thread
///////////////// Start Server Balance Thread
///////////////// Start MainServer Response Thread
///////////////// Start SubServer-1 Response Thread
///////////////// Start SubServer-2 Response Thread
///////////////// Start List Thread
///////////////// Start capatityControlThread Thread
///////////////// Start closeControlThread Thread
**********************************************************************

MainServer      10000      %100.0
SubServer-1      796      %15.92
SubServer-2      456      %9.12
Çalışan Thread Sayısı: 9
Çalışan Sunucu Sayısı: 3


**********************************************************************
**********************************************************************

MainServer      9855      %98.55
SubServer-1      726      %14.52
SubServer-2      330      %6.6
Çalışan Thread Sayısı: 9
Çalışan Sunucu Sayısı: 3


**********************************************************************
**********************************************************************
```

```
**********************************************************************
**********************************************************************

MainServer      10000      %100.0
SubServer-1      3370      %67.4
SubServer-2      1241      %24.82
SubServer-5      908      %18.16
Çalışan Thread Sayısı: 10
Çalışan Sunucu Sayısı: 4


**********************************************************************
------------BÖLÜNME İŞLEMİNDEN ÖNCE------------
------------------------------>DivideServer
------------------------------>divideRequest
------------------------------>CreateNewServer
///////////////// Start SubServer-6 Response Thread
------------BÖLÜNME İŞLEMİNDEN SONRA------------
**********************************************************************

MainServer      9968      %99.68
SubServer-1      1903      %38.06
SubServer-2      1433      %28.66
SubServer-5      485      %9.7
SubServer-6      2104      %42.08
Çalışan Thread Sayısı: 11
Çalışan Sunucu Sayısı: 5


**********************************************************************
**********************************************************************
```

```
**********************************************************************

Request Bekliyor

**********************************************************************


MainServer      0      %0.0

SubServer-1      0      %0.0

SubServer-2      0      %0.0

Çalışan Thread Sayısı: 9

Çalışan Sunucu Sayısı: 3



**********************************************************************

Request Bekliyor

**********************************************************************
```

```
**********************************************************************
**********************************************************************

MainServer      10000      %100.0
SubServer-1      1979      %39.58
SubServer-2      2928      %58.56
SubServer-4      195      %3.9
Çalışan Thread Sayısı: 10
Çalışan Sunucu Sayısı: 4


**********************************************************************
SubServer-4Response Thread Stop
---------------->Stop etti
---->DELETESubServer-4    0
**********************************************************************

MainServer      9909      %99.09
SubServer-1      1783      %35.66
SubServer-2      2684      %53.68
Çalışan Thread Sayısı: 9
Çalışan Sunucu Sayısı: 3


**********************************************************************
**********************************************************************
```

VI. Kazanımlar

Sunucu İstek Yoğunluğunun Multithread İle Kontrolü Projesinin bize kantıkları;

- Threadler yardımıyla birden fazla işin bir arada nasıl yapılacağını öğrendik.
- Birden fazla threadin nasıl yönetileceğini ve aynı alana erişimi sırasında karşılaşılan eş zamanlılık hatalarının nasıl önüne geçiceleceği öğrendik
- Main ve Sub Serveri ana sunucudan extends ederek oop mantığındaki eksiklerimizi tamamladık.
- Bilgisayarın birden fazla işi nasıl yaptığı hakkında daha fazla fikir sahibi olduk

VII . Karşılaşılan Sorunlar Ve Çözüm Yöntemleri

1. concurrentmodificationexception request datalarımıza bir şeyler eklerken aynı zamanda çıkartmak istediğimizde eş zamanlılık hatası aldık bu tip problemlerin önünce geçmek için locklar kullandık bu sayede bir ekleme yapılırken aynı zamanda çıkarma işlemi yapılmıyor
2. illegalmonitorstateexception threadleri listeden stop işlemi ile durduğumuzda bu hata ile kaşılaştık hatanın sebebi stop işemini yanlış yerde yapmamızdı.
3. createnewobje problemini lockları kullanırken yaşadık lockları wait fonkisyonu ile beklemeye alırken yaptığımız yanlış kullanımı düzelttik
4. Oluşduğumuz sunucuları yönetme konsunda problem yaşadık bu problemi listelerden yararladık.Bu şekilde sunucularınımız yönetimi kolaylaştı
5. java.lang.arrayindexoutofboundsexceptio n datalarımızı listelerin içinde tuttuğumuz için bol bol bu hata ile karşılaştık bazı serverlede silme işlemi için foreach döngüsünden yararlanmıştık burada sublist kullanımı bizim için problem yaratmıştı.Bunu çözmek için klasik for kullanımını gerçekleştirdik.

Kaynakça

[1] **Java ile Thread Kullanımı**
https://cse.iitkgp.ac.in/~dsamanta/java/ch6.htm
https://ufukuzun.wordpress.com/yayinlarim/javada-multithreading/
https://dspace.mit.edu/bitstream/handle/1721.1/80050/43441068-MIT.pdf?sequence=2
https://www.callicoder.com/java-executor-service-and-thread-pool-tutorial/
Mustafa Murat Çoşkun Udemy Kursu

[2] **Task parallelism ve LoadBalance**

https://winterbe.com/posts/2015/04/07/java8-concurrency-tutorial-thread-executor-examples/
https://stackoverflow.com/questions/2016083/what-is-the-easiest-way-to-parallelize-a-task-in-java
https://medium.com/@wolfbang/load-balance-algorithm-with-java-e7fb55fe788a

[3] **Karşılaşınlaşılan Hataların Kaynakları**

Concurrentmodificationexception
https://airbrake.io/blog/java-exception-handling/concurrentmodificationexception
https://stackoverflow.com/questions/13133688/java-util-concurrentmodificationexception-when-adding-another-object
https://stackoverflow.com/questions/49971932/java-util-concurrentmodificationexception-thrown-when-adding-to-list

İllegalmonitorstateexception
https://stackoverflow.com/questions/47730181/how-to-make-a-thread-wait-until-an-array-contains-elements

createnewobje problem
https://stackoverflow.com/questions/17840397/concept-behind-putting-wait-notify-methods-in-object-class

Karşılaşılan diğer problemlerin çözümünü yukardaki kaynaklardan karşıladık.