SUNUCU İSTEK YOĞUNLUĞUNUN MULTITHREAD İLE KONTROLÜ

Muhammed Emre KARA

160202094

Bilgisayar Mühendisliği Bölümü
Kocaeli Üniversitesi

mailemrek@gmail.com

1. Özet

Bu projede sunucu istek(server-client) yoğunluğu problemine multithreading kullanılarak çözümler üretilmesi beklenmektedir. Sunucular gerçek birer fiziksel sunucu olmayacak sadece bu istek – cevap yapısını simule edecektir.

Bu işlemi gerçekleştirmek için JAVA proglama dili tercih edildi. Sonuçlar ise Java-Swing ile hazırlanmış bir arayüz aracılığıyla sunuldu.

2. Proje bileşenlerinin özellikleri:

2.1 Ana Sunucu (Main Thread): 10000 istek kapasitesine sahiptir. 500 ms zaman aralıklarıyla [1-100] arasında rastgele sayıda istek kabul etmektedir. İstek olduğu sürece 200 ms zaman aralıklarıyla [1-50] arasında rastgele sayıda isteğe geri dönüş yapmaktadır

2.2 Alt Sunucu (Sub Thread): 5000 istek kapasitesine sahiptir. 500 ms zaman aralıklarıyla [1-50] arasında rastgele sayıda ana sunucudan istek almaktadır. İstek olduğu sürece 300 ms zaman aralıklarıyla [1-50] arasında rastgele sayıda isteğe geri dönüş yapmaktadır

2.3 Alt sunucu oluşturucu (Sub Thread): Mevcut olan alt sunucuları kontrol eder. Eğer herhangi bir alt sunucunun kapasitesi %70 ve üzerinde ise yeni bir alt sunucu oluşturur ve kapasitenin yarısını yeni oluşturduğu alt sunucuya gönderir. Eğer herhangi bir alt sunucunun kapasitesi %0 a ulaşır ise mevcut olan alt sunucu sistemden kaldırılır. En az iki alt sunucu sürekli çalışır durumda kalması gerekmektedir

2.4 Sunucu takip (Sub Thread): Sistemde mevcut olan tüm sunucuların bilgilerini (Sunucu/Thread sayısı, ve kapasiteleri(%)) canlı olarak göstermektedir.

3. Temel Bilgiler

Projeyi gerçekleştirirken kullandığımız teknolojiler aşağıda verilmiştir.

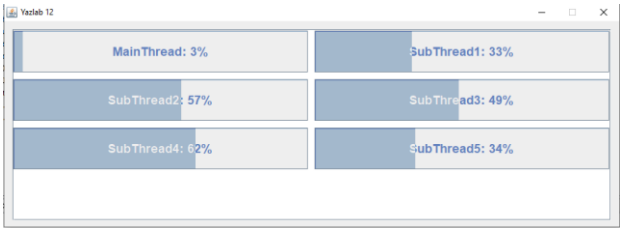
JAVA: Java, Sun Microsystems mühendislerinden James Gosling tarafından geliştirilmeye başlanmış açık kodlu, nesneye yönelik, zeminden bağımsız, yüksek verimli, çok işlevli, yüksek seviye, adım adım işletilen (yorumlanan-interpreted) bir dildir.

SWING: Swing kütüphanesi Sun Microsystems tarafından piyasaya sürülen resmi Java GUI aracıdır. Swing; %100 java ile yazılmıştır ve Java Foundation Classes(JFC) nin bir parçasıdır. Tam donanımlı masaüstü uygulamaları geliştirmek için kullanılan paketler bütünüdür. JFC; AWT ,Swing ,Ulaşılabilirlik,Java 2D ve Sürükle Bıraktan oluşur. 1997 yılında JDK 1.2 ile birlikte piyasaya sürülmüştür.

Benim Projemde:

SystemInterface:

Kullanıcı arayüzü için java JFrame sınıfıdır. Tüm GUI işlemleri burada bulunur. Tasarımı aşağıda gösterilen şekildedir:



Var olan tüm threadlerin yani sunucuların kapasitelerini yüzdelik olarak gerçek zamanlı gösteren bir ekran görevindedir. Yeni eklenen threadler eklenirken silinen threadler ise ekrandan kaldırılır.

Yazlab212:

Başlatıcı sınıftır. Diğer tüm sınıflar bu sınıf içerisinden çağrılır ve sistem başlatılır.

MainThread:

Anasunucuyu temsil eden sınıftır. 2 tane alt Thread barındırır bunlar:

1. **GetRequestMainThread:** Ana sunucunun rsatgele istek kabul etme fonksiyonunu gerçekleştirir. jobList Kuyruğuna yükleme yapar
2. **ServeMainThread:** Ana sunucunun rastgele istek yanıtlama fonksyionunu gerçekleştirir. jobList kuyruğunu boşaltır.

jobList: Ana sunucunun istek 10.000 kapasiteli istek havuzudur. Queue (Kuyruk Tipindedir)

```
synchronized (jobList) {
    try {
        for (int i = 0; i < randomValue; i++) {

            try {
                jobList.remove();
            } catch (Exception e) {
            }
        }
```

SubThread:

Alt Sunucuları temsil eden sınıftır. 2 tane alt Thread barındırır bunlar:

1. **GetRequestSubThread: Alt sunucunun rsatgele istek sayıda Anasunucunun jobList kuyruğundan istek alma fonksiyonunu gerçekleştirir. subJobList Kuyruğuna yükleme yapar**
2. **ServeSubThread: Alt sunucunun rastgele istek yanıtlama fonksyionunu gerçekleştirir. subJobList kuyruğunu boşaltır.**

subJobList: Her alt sunucunun kendisine ait olan 5.000 kapasiteli istek havuzudur. Queue (Kuyruk Tipindedir)

GetRequest:

```
synchronized (subJobList) {
    synchronized (jobList) {
        for (int i = 0; i < randomValue; i++) {
            try {
                subJobList.add(jobList.remove());
            } catch (Exception e) {
            }
        }
    }
}
```

Serve:

```
while (true) {
    try {
        Thread.sleep(300);
    } catch (InterruptedException ex) {
        Logger.getLogger(ServeSubThread.class.getName()).log(Level.SEVERE, null, ex);
    }
    int randomValue = rand.nextInt(50) + 1;
    //System.out.println("\nistek sayisi: " + randomValue);
    try {
        for (int i = 0; i < randomValue; i++) {
            synchronized (subJobList) {
                subJobList.remove();
            }
        }
    } catch (Exception e) {
        //System.out.println("Queue is Empty!");
    }
    //System.out.println("istekler yanitlandi..\n");
    //System.out.println(this.getName() + ": " + subJobList.size());
}
```

SubThreadCreator:

Alt Sunucuları oluşturan sınıftır. Kendisi bir Thread olarak çalışır, Anasunucu yoğunluğunu ve Altsunucuların yoğunluklarını kontrol eder. 3 temel veri tipi barındırır bunlar:

1. **HashMap progressBars:** SystemInterface'de kullanıcıya gösterilecek **JProgressBar'ları** içerir**.**
2. **Queue jobList:** Anasunucunun jobList'ini refer eder.
3. **CopyOnWriteList tgr**: Alt sunucuları tutan bir çeşit **ArrayList**'dir. Sıradan bir ArrayList'den farkı iterasyon esnasında listede değişikliğe izin vermesidir. Bu sebeple multithread uygulamalarda sıkça kullanılır.

Başlangıçta 2 tane SubThread yani AltSunucu oluşturur ve her zaman en az 2 tanesini korur. **900ms** de bir kontrollerini yapar. %70'in üzerinde kapasiteye sahip alt sunucuların yükünü yeni bir alt sunucu oluşturarak böler.

Eğer kapasitesi 0'a ulaşmış bir Altsunucu varsa o alt sunucuyu kapatır ve progressbarını listeden siler.

```
if (percentage > 70) {
    counter++;
    tgr.add(new SubThread(this.jobList, ("SubThread" + counter)));
    System.out.println("Pre Adding new Thread! => " + s.getName() + ": " + s.
    synchronized (s.subJobList) {
        for (int i = 0; i < s.subJobList.size(); i++) {
              String pre = "  pre: " + s.subJobList.size() + " ," + tgr.get(c

            //System.out.print("  pre: " + s.subJobList.size() + " ," + tgr.g
            try {
                tgr.get(counter - 1).subJobList.add(s.subJobList.remove());
            } catch (Exception e) {
                System.out.println("counter: " + counter);

                tgr.get(counter-1);
                System.out.println("i: " + i);
                System.out.println("  to post: " + s.subJobList.size() + " ,"
                System.exit(0);
            }

            //System.out.println("  to post: " + s.subJobList.size() + " ," +
        }
    }
    progressBars.put(("SubThread" + counter), new myProgressBar(("SubThread" +
    progressBars.get(("SubThread" + counter)).set4152((int) percentage);
    System.out.println("Add new Thread! => " + s.getName() + ": " + s.subJobL
    tgr.get(counter - 1).start();
} else if (percentage == 0 && tgr.size() > 2) {
    System.out.println(s.getName() + " removed." + " size= " + s.subJobList.s
    s.interrupt();
    progressBars.remove(s.getName());
    //tgr.remove(s);
}
```

Kaynakça

1) http://javayaz.com/?page_id=107
2) https://www.javatpoint.com/multithreading-in-java /
3) https://stackoverflow.com/questions/671049/how-do-you-kill-a-thread-in-java/
4) https://www.callicoder.com/java-multithreading-thread-and-runnable-tutorial/
5) https://www.geeksforgeeks.org/copyonwritearraylist-in-java//