BABEŞ-BOLYAI UNIVERSITY CLUJ–NAPOCA
FACULTY OF MATHEMATICS AND INFORMATICS
SPECIALIZATION: COMPUTER SCIENCE

Dissertation

Javascript based grid computing

SCIENTIFIC SUPERVISOR:
DR.BODÓ ZALÁN, LECTURER

STUDENT:
ISZLAI LEHEL ISTVÁN

JUNE 2014

UNIVERSITATEA BABEŞ-BOLYAI, CLUJ–NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

Lucrare de licenţă

Grid computing folosind Javascript

CONDUCĂTOR ŞTIINŢIFIC:
DR.BODÓ ZALÁN, LECTOR UNIVERSITAR

ABSOLVENT:
ISZLAI LEHEL ISTVÁN

IUNIE 2014

BABEŞ-BOLYAI TUDOMÁNYEGYETEM KOLOZSVÁR
MATEMATIKA ÉS INFORMATIKA KAR
INFORMATIKA SZAK

Szakdolgozat

Javascript alapú elosztott számítási rendszer

TÉMAVEZETŐ:
DR.BODÓ ZALÁN, ADJUNKTUS

SZERZŐ:
ISZLAI LEHEL ISTVÁN

2014 JÚNIUS

BABEŞ-BOLYAI UNIVERSITY CLUJ–NAPOCA
FACULTY OF MATHEMATICS AND INFORMATICS
SPECIALIZATION: COMPUTER SCIENCE

Dissertation

Javascript based grid computing

Abstract

The power of grid computing has helped many scientists to achieve the same computational power of large-scale mainframes. There are a few open source desktop grid computing frameworks like BOINC (Berkeley Open Source) or Folding@home, but these require the installation of client applications that often need to be ported to different operating systems. The scope of this paper is to try to recreate a similar system based on Javascript, so it can leverage the platform independence of the web. The framework created is named Youshare and it is built with the latest web technologies available.

This work is the result of my own activity. I have neither given nor received unauthorized assistance on this work.

2014 JUNE

ISZLAI LEHEL ISTVÁN

ADVISOR:
DR.BODÓ ZALÁN, LECTURER

Tartalomjegyzék

1. Bevezető **4**

2. Felhasznált technológiák, eszközök és módszerek **6**
2.1. Play framework 6
2.1.1. A build rendszer 6
2.1.2. H2 adatbázis 7
2.1.3. Adatbázis evolúciós szkriptek 7
2.1.4. Play framework-ben írt alkalmazás felépítése 8
2.1.5. Egy kérés életciklusa 9
2.2. Apache ActiveMQ 10
2.3. AngularJS keretrendszer 11
2.3.1. Más Javascript technológiák 12

3. A Youshare projekt követelményei **14**

4. Elemzés és tervezés **16**
4.1. Használati esetek 16
4.2. A rendszer architektúrája 17

5. Megvalósítás **19**
5.1. A szerver implementációja 19
5.1.1. Az adatbázis és adatmodell implementációja 19
5.1.2. Az MQ implementációja 20
5.1.3. A Controllerek implementációja 21
5.2. A kliens implementációja 23
5.2.1. Az app.js implementációja 23
5.2.2. A servicek implementációja 24
5.2.3. A kontrollerek és nézetek implementációja 25

6. Mérési adatok **27**
6.1. Apache JMeter 27

6.2. A mérések részletezése 27

7. Következtetések és további fejlesztési lehetőségek 30

1. fejezet

Bevezető

A dolgozat Javascript alapú elosztott számítási rendszeralkalmazások megvalósítását vizsgálja, ezáltal egyszerűbbé téve a kisebb saját elosztott számítási rendszerek létrehozását és kiépítését. A projekt a régi natív kliens alapú elosztott számítási rendszerekhez nyújt egy alternatívát, egy könnyűsúlyú webalkalmazás formájában. Keretrendszerek bemutatását foglalja magába, melyek lehetőséget nyújtanak modern webalkalmazások fejlesztésére.

A projekt neve Youshare, egy alkalmazás amely elosztott számítási rendszerek teljesítményét teszi elérhetővé bárki számára egy szociális hálóra építve. A célja egy olyan alkalmazás létrehozása, amely segítségével bárki létre hozhat önmagának egy elosztott számítási rendszert a saját feladataival, majd megosztja ezt a problémát másokkal, akik böngészőn keresztül becsatlakozhatnak az elosztott számítási rendszerbe a saját erőforrásukat felajánlva.

Az adott probléma teljesen a elosztott számítási rendszer tulajdonosa által megadható Javascript nyelven van programozva. Ha szükséges, akkor az elosztott számítási rendszer tulajdonosa, adatszeleteket is megoszthat CSV[1] fájl formátumban feltöltve.

A klienseknek másra nincs szüksége, csak egy böngészőben felkeresni az elosztott számítási rendszer tulajdonosa által megosztott linket. A megadott helyen a böngésző elkezdi letölteni a kliensnek kiosztott adat szeletet, majd futtatni kezdi a kiosztott feladatot, így elvégezve a saját feladatát a hálón belül.

A dolgozatban bemutatott Youshare rendszer egy prototípusnak tekinthető, mely egy hasonló rendszer megvalósíthatóságát és használhatóságát próbálja felmérni.

A dolgozat első részében bemutatásra kerülnek a fontosabb technológiák

[1] `http://en.wikipedia.org/wiki/Comma-separated_values`

és eszközök, amelyek a Youshare megvalósításához elengedhetetlenek voltak.

A második részében a projekt részletesebb bemutatása, annak funkcionális és nem funkcionális követelményei olvashatóak.

A harmadik rész bemutatja a rendszer követelményeinek elemzését, a használati eseteit, valamint a rendszer architektúráját.

A negyedik részben a projekt konkrét megvalósításának folyamatai és elemei lesznek bemutatva a kliens és a szerver oldaláról egyaránt.

Az ötödik részben az elkészült projekt teljesítményével kapcsolatos mérések eredményei, és azok elemzése tekinthető meg.

A hatodik részben olvasható néhány következtetés és továbbfejlesztési lehetőség a Youshare projekttel kapcsolatosan.

2. fejezet

Felhasznált technológiák, eszközök és módszerek

2.1. Play framework

A Play framework[1] egy olyan nyílt forráskódú keretrendszer, amely lehetővé teszi webalkalmazások fejlesztését Java és Scala[2] programozási nyelven. Könnyen integrálható más modern keretrendszerekkel, melyek szükségesek a modern webalkalmazások fejlesztéséhez.

A Play framework aszinkron programozási modellre épít. Ezáltal fel van készülve, hogy bármely HTTP kérés hosszú élettartamú lehet. Mindezt úgy éri el, hogy kihasználja a híres actor-model alapú konkurens programozási keretrendszert, az Akka-t[3].

Natív támogatást nyújt úgy a Java, mint a Scala programozási nyelvhez, így egyszerűen keverhető a Java ipari képességű stabil platformja és a Scala által nyújtót funkcionális programozás előnyei.

2.1.1. A build rendszer

A Play framework az SBT nevű *build* rendszerre alapszik, mely egy nyílt forráskódú Scala és Java build rendszer. Hasonló az ismert Maven és Ant rendszerekhez. A build folyamat leírható Scala-ban vagy a saját DSL[4]-jében. Az SBT a függőségek kezelésére az Ivy rendszert használja. Az Ivy rendszer a saját katalógusain kívül támogatja a Maven katalógusokat is, ezáltal könnyen integrálható más projektekkel.

[1] `http://www.playframework.com/`
[2] `http://www.scala-lang.org`
[3] `http://akka.io`
[4] `http://en.wikipedia.org/wiki/Domain_Specific_Language`

2.1.2. H2 adatbázis

A Play framework alapértelmezetten egy H2[5] adatbázis-kezelő rendszert használ. Rengeteg előnye van, melyek közül az alábbiak a fontosabbak:

- Gyors a kis méretű adatok tárolására
- Nyílt forráskódú és elérhető jdbc API-n keresztül
- Lehet használni egyaránt *in-memory* (csak memóriában tárolt) és fájl alapú perzisztens adattárolás is
- Kis méretű és kevés függőséggel rendelkezik (az egész csomag kisebb mint 1.5 MB)

A H2 adatbázis-kezelő rendszer a Play framework-on belül elérhető több ORM[6]-en keresztül is például a JPA alapú Ebean -en és Anorm-on.

2.1.3. Adatbázis evolúciós szkriptek

A Play framework számontartja az adatbázis séma változásait, úgynevezett evolúciós szkriptek segítségével. Egy ilyen evolúciós szkript valójában egyszerű SQL, mely számontartja, hogy egy evolúció hogyan érhető el az "!Ups" részben, és hogyan fordítható vissza a "!Downs" részben a különböző séma állapotban.

Ezáltal bármikor újra létrehozható az adatbázis séma, ami gyakran előfordul ha a H2 adatbázis-kezelő rendszer in-memory módban van használva.

Példa egy evolúciós szkriptre:

```
# Users schema
# -- !Ups

CREATE TABLE User (
    email varchar(255) NOT NULL,
    password varchar(255) NOT NULL,
    fullname varchar(255) NOT NULL,
    isAdmin boolean NOT NULL,
    PRIMARY KEY (id)
);

# -- !Downs

DROP TABLE User;
```

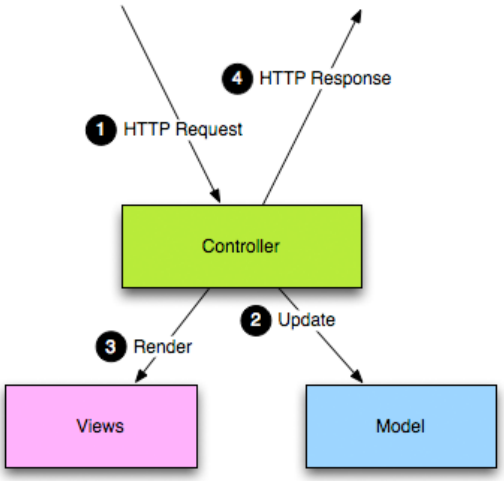
[5] http://www.h2database.com/html/main.html
[6] http://www.orm.net

2.1.4. Play framework-ben írt alkalmazás felépítése

A Play framework követi a *convetion over configuration* elvet, lényege, hogy a szabványt részesíti előnyben a különböző konfigurációkkal/konfigurálással szemben. Mivel a Play framework-ben írt alkalmazások MVC (Model View Controller) mintára épülnek ez meglátszik az alkalmazás struktúráján is.

```
app/controllers
app/models
app/views
```

MVC elvet követve a 2.1 ábra jól láthatóan felvázolja, hogyan állnak ezek az entitások egymással kapcsolatban egy webalkalmazás keretein belül.



2.1. ábra. Play MVC bemutatása
Forrás: `http://www.playframework.com/documentation/1.2.7/main`

Az app mappa mellet fontosak még a következő elemek, melyek szerves részét képezik egy Play framework-ben írt alkalmazásnak.

A **public/** mappába kerülnek az olyan htm/css/javascript/... fájlok, melyek bárki számára elérhetőek lesznek.

A **conf/** mappába kerülnek az **application.conf** és **routes** fájlok, ahol az application.conf tartalmazza a naplózást, adatbázis elérést és hasonló konfigurációkat, valamint a routes és a HTTP kérések útvonal alapú irányítását a megfelelő kontroller metódusokhoz.

A **lib/** mappába megadhatóak olyan jar függőségek, melyek nem a build rendszer alapértelmezett függőségkezelő rendszere (Ivy) által vannak betöltve.

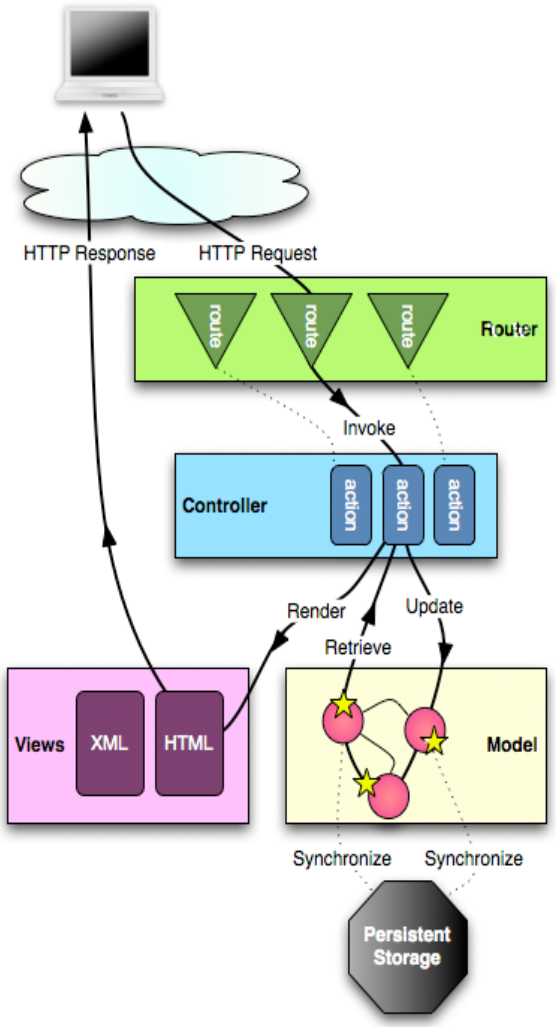
A **build.sbt** file tartalmazza a build folyamat leírását, ezt felhasználva készíti el a fent említett SBT a végső build-et.

2.1.5. Egy kérés életciklusa

A Play framwork-ben egy kérés 5 egyszerű lépésen valósul meg:

1. A keretrendszerbe beérkezik egy HTTP kérés
2. Az alkalmazás routere megpróbálja a megfelelő kontroller metódushoz irányítani a fent említett router konfigurációs file alapján
3. Az alkalmazás kódja végrehajtódik
4. Ha több nézetre van szükség, akkor létrejön egy összetett nézet
5. A megfelelő HTTP válaszba beírja a visszatérített nézetet és a válasz paramétereket

Mindez megtekinthető a 2.2 ábrán.



2.2. ábra. Egy kérés életciklusa
Forrás: `http://www.playframework.com/documentation/1.2.7/main`

2.2. Apache ActiveMQ

Az Apache ActiveMQ[7] egy üzenetküldő keretrendszer, mely teljes mértekben implementálja a JMS (Java Message Service) specifikációját. Az Apache ActiveMQ a Java nyelv mellet natívan használható más nyelvekből is, ilyen

[7] `http://activemq.apache.org/`

például a C, C++, C♯, Ruby, Perl, Python és PHP. Emellett bármilyen más rendszerből is használható az általa nyújtott REST API-n keresztül.

Támogatja a következő szállítási protokollokat: in-VM, TCP, SSL, NIO, UDP, multicast, JGroups, JXTA. JDBC-n keresztüli gyors perzisztenciára képes, de a legfontosabb tulajdonságai a projekt szempontjából, hogy képes tranziens, perzisztens (kitartó) és tranzakciós üzenetküldésre.

2.3. AngularJS keretrendszer

Az AngularJS egy strukturális keretrendszer dinamikus kliens-oldali web-alkalmazásokhoz. A megjelenítést teljesen HTML sablonokra építi, de ezek bármikor bővíthetőek direktívák létrehozásával.

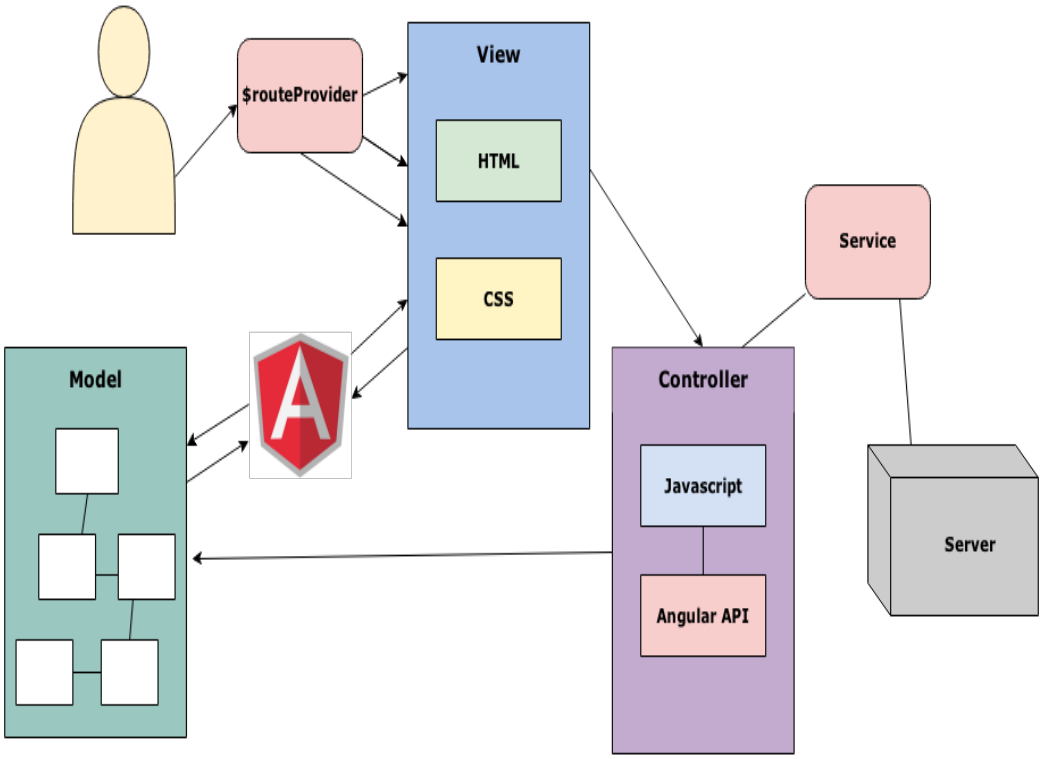
A direktívák lehetőséget nyújtanak számunkra, hogy saját HTML5 tageket hozzunk létre, melyek megjelenítését mi magunk definiáljuk. Az AngularJS támogat egy *dependecy injection* (függőségi befecskendezés, kezelés) rendszert is, ezáltal lehetővé téve külső függőségek megadását, példányosítását és újrahasználhatóságát.

Ezen kívül egy nagyon erős és flexibilis *two-way data binding* (kétirányú adatkötést) is támogat, melyet kihasználva az adat változását dinamikusan a megjelenítésben is változik, és fordítva a megjelenítésben végzett változtatás közvetlenül kihat az adatmodellre is.

Az AngularJS egy Model View Whatever (Model Nézet Mindegy) keretrendszernek nevezi magát, mert az általa nyújtott funkcionalitás megfelelő az alábbi megjelenítési, tervezési minták bármelyikére:

- MVC - Model-View-Controller
- Model-View-Presenter
- MVVW - Model-View-ViewModel
- HMVC - Hierarchical Model-View-Controller
- MMV - Multiuse Model View
- MVA - Model-View-Adapter

Egy AngularJS alkalmazás tipikusan az MVC elvet követi, ahol a modell és a nézet között egy kétirányú adatköteg található. A kontroller a szerver oldallal egy teljesen RESTful (Representational State Transfer) kommunikációt használ. Mindez megtekinthető a 2.3 ábrán.



2.3. ábra. Tipikus AngularJS alkalmazás

2.3.1. Más Javascript technológiák

A**Grunt** egy Javascript feladatfuttató. Rengeteg ismétlődő és monoton munkát lehet vele megspórolni, mert benne definiálhatunk feladatokat, például Javascript fájlok összefűzése, tömörítése vagy akár unit tesztek futtatása. Egyszerűen bővíthető a funkcionalitása a nagy mennyiségű hozzá készített plug-in-ek (kiegészítő modulok) felhasználásával. Használatához két fájlra van szükség: a *package.json*, ami a grunt projekt tulajdonságait és kiegészítőit tartalmazza, és a *Gruntfile*, ami a feladatok definiálására ad lehetőséget. NodeJS-re épül így szükséges a telepítése a Grunt használatához. A NodeJS telepítése után a *npm install -g grunt* paranccsal telepíthetjük a Grunt-ot is.

A **Bower** egy csomag-, és függőségkezelő a Javascript-hez. Hasonló a Javas Maven vagy Ivy-hez. Megkönnyíti a Javascript modulok és függőségek beszerzését és verziókezelését. Használatához csupán két fájlra van szükség: a *.bowerrc*, ahol definiáljuk, hogy hova töltse a függőségeket, és a *bower.json*, ahol megadjuk az általunk igényelt modulokat. Szintén NodeJS-re épül és a *npm install -g bower* paranccsal telepíthető.

A **Yeoman** modern webalkalmazások megépítéséhez nyújt segítséget. Scaffolding (állványozó rendszer), mely segít új webalkalmazások kezdeti lépéseiben. Kigenerál bizonyos gyakran használt fájlokat. Létrehozza a szükséges fent említett Grunt és Bower fájlokat, és egy kezdeti projektstruktúrát, amely a használt technológiától függő. Legismertebb generátorai az AngularJS, BackboneJS és EmberJS keretrendszerekhez alkalmazhatóak, de emellett jelenleg több mint 700 másik keretrendszerhez használható.

Az **Ace Editor** egy webes nyílt forráskódú szövegszerkesztő, mely könnyen integrálható bármilyen webalkalmazásba. Teljesen Javascript-ben van írva, de teljesítménye hasonló bármilyen natív szövegszerkesztőhöz, amilyen a Vim vagy az Emacs. Főbb karakterisztikái:

- több mind 100 programozási nyelvet támogat szintaxiskiemeléssel
- szintaxis ellenőrzést támogat
- több mind 20 szín témával rendelkezik
- támogatja a Vim és Emacs billentyűzet rövidítéseit
- támogatja a hatalmas fájlok megnyitását (kb. 4 millió sor)

A **JSON** avagy (JavaScript Object Notation)[8] , egy adatcserére létrehozott szabvány. Könnyen olvasható ember számára, de programkódból is könnyen kiolvasható parser-ket felhasználva. A szabványt alkalmazva könnyedén reprezentálhatóak az egyszerű adatstruktúrák és asszociatív tömbök.

A JSON fájlok kiterjesztése a *.json*, de legtöbbször kliens és szerver közti adatátvitelére használatos, ahol a hivatalos MIME-típusa az *application/json*.

Javascriptben használható elsőosztályú adatstruktúraként, míg más programozási nyelvekhez egyszerűen találhatóak átalakítok, melyek a nyelv objektumait JSON-be és vissza tudják alakítani. Ilyen például a Java-ban a Jackson keretrendszer.

[8] `http://json.org/`

3. fejezet

A Youshare projekt követelményei

A Youshare elosztott számítási rendszert biztosít, melyet bárki egyszerűen üzemeltethet különösebb költségek vagy speciális hardver nélkül önkéntesek segítségét felhasználva.

Az elosztott számítási rendszer üzemeltetője és vagy megbízottjai a projektje leírásával népszerűsítheti a saját projektjét a felhasználok között, ezáltal növelve a háló teljesítményét. A felhasználóknak lehetőségük van keresni olyan projektet melyet támogatnak, és a böngészőjüket használva segítséget nyújthatnak új kutatásoknak, úgymond feláldozva a saját számítógépüket mikor épp nem használják azt vagy nélkülözhető számukra.

A szerver funkcionális követelményei:

- Adatok tárolása
- Kliensekkel való kommunikáció
- Felhasználók adatainak ellenőrzése bejelentkezés esetén
- MQ feltöltése adatokkal
- Kliensek kiszolgálása az MQ-ból kinyert adatokkal

A kliens funkcionális követelményei:

- A üzemeltetője, illetve üzemeltetői autentikációja
- A projekt leírásának létrehozása illetve szerkesztésére felület biztosítása
- Az elosztott számítási rendszer programozására, feladat definiálására egy modern szövegszerkesztő biztosítása

- Az elosztott számítási rendszer feladatához szűkséges adatok feltöltésének kezelése
- Önkéntesek becsatlakoztatása a megfelelő hálóba

Nem funkcionális követelmények:

- A szerver platformfüggetlensége
- Egyszerű telepítési és indítási folyamat
- Hardverigény minimalizálása nagy mennyiségű kapcsolat esetén is
- A terhelés nagyobb arányban legyen kliens oldalon a szervernek kedvezve
- A kliens oldal legyen felhasználóbarát
- Támogassa a mobil és a nagy felbontású böngészőket egyaránt

4. fejezet

Elemzés és tervezés

Az elemzés során fontosabb használati esetek, a rendszer architektúrája kerül részletesebb bemutatásra. Az egyes magyarázatok mellé szemléltető ábrákat is csatoltunk, melyek segítik a rendszer és komponensei közti kapcsolatok megértését.

4.1. Használati esetek

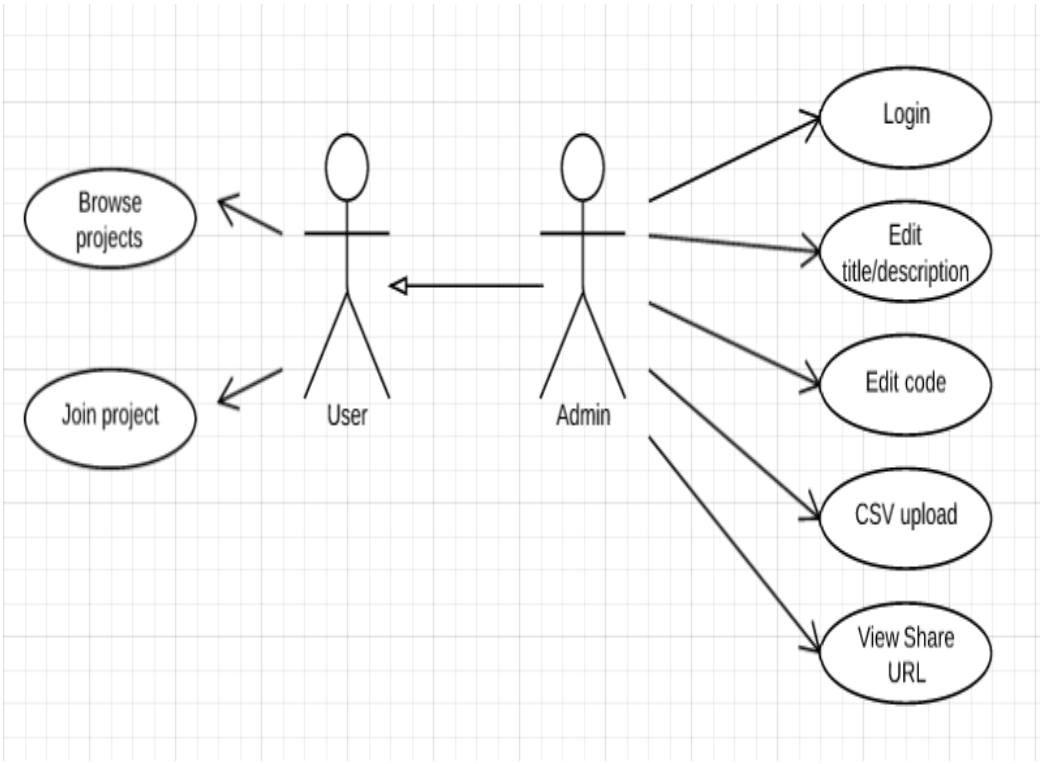
Az elosztott számítási rendszer üzemeltetőjének és a felhatalmazott felhasználóknak a következő használati esetek vannak biztosítva:

- Bejelentkezés
- Projekt címének és leírásának szerkesztése
- Projekt feladatának szerkesztése
- Projekt feladatához szükséges CSV adatfájl feltöltése
- Kész projekt elérési adatainak megtekintése

Az oldal nem felhatalmazott látogatói által elérhető használati esetek:

- Projektek böngészése
- Projektekhez való csatlakozás

A fent említett használati eseteket bemutatja a 4.1 ábra.



4.1. ábra. Use-case diagram

4.2. A rendszer architektúrája

A rendszer architektúrája, mint ahogy a 4.2 ábrán is látható két főbb részre bontható: a már említett play framework által biztosított szerver alkalmazásra és az AngularJS által biztosított kliens oldali komponensre. A két fő rész mellet még megjelenik az ActiveMQ komponens, mely segíti a szerver oldalt az adatszeletek kiszolgálásával és tranzakció-kezeléssel.

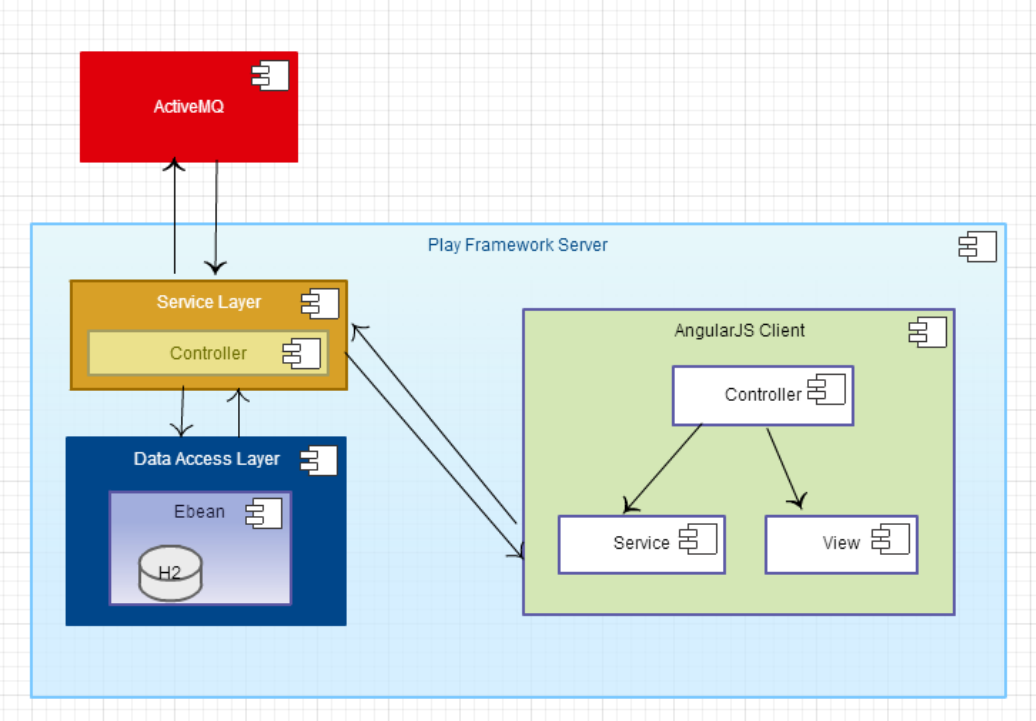
A kliens oldali AngularJS komponens a Play framework által nyújtott HTTP szerver segítségével lesz elérhető, de utána a kliens oldalon különálló folyamatként fut és teljesen RESTful[1] módon kommunikál a továbbiakban a szerver alkalmazással. Az adatok a hálózaton keresztül JSON formátumban vannak továbbítva.

A szerver oldal az ActiveMQ-val a JMS[2]-en keresztül egy üzenetküldési mechanizmuson alapuló kommunikáción megy keresztül.

[1] `http://hu.wikipedia.org/wiki/REST`

[2] `http://docs.oracle.com/cd/E19957-01/816-5904-10/816-5904-10.pdf`

Az adatok egy H2 adatbázisban vannak tárolva, melyhez a szerver oldal egy Ebean[3] objektum-relációs leképzésen alapuló keretrendszert használ.



4.2. ábra. Architektúra diagram

[3] http://www.playframework.com/documentation/2.2.0/JavaEbean

5. fejezet

Megvalósítás

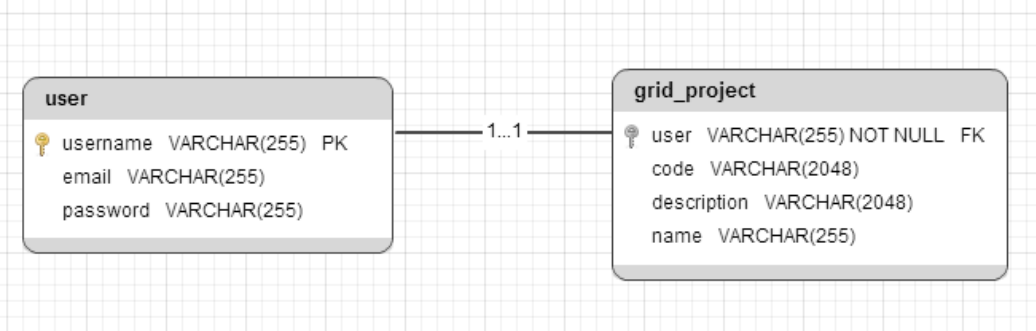
Amint az már az előzőekben olvasható volt, bemutatásra került a YouShare alkalmazás által használt különböző technológiák és elméleti háttér, valamint a követelmények és a hozzá készített tervek. Mindezek szükségesek voltak a megvalósítás megértéséhez. A dolgozat hátralevő részében bemutatásra kerül a megvalósítás részletes leírása valamint a programozás során felmerült akadályok, és az azokhoz talált megoldások.

5.1. A szerver implementációja

A szerver, az alkalmazás keretein belül, főként az adatok irányításáért és a kliensek kiszolgálásáért felelős. Ő tartja a kapcsolatot a H2 adatbázissal, valamit az Apache ActiveMQ-val, és a megfelelő adatokat továbbítja a kliensek felé. A szerver implementációja során a legfőbb szerepet az MVC (Modell Nézet Vezérlő) tervezési minta kapja, mivel a Play framework keretrendszer is erre épül.

5.1.1. Az adatbázis és adatmodell implementációja

A szerver tartja a kapcsolatot a H2 adatbázissal, de a tárolt adatok mennyisége csekély, mivel a fő cél az volt, hogy a szerver oldali erőforrásszükséglet a lehető legkisebb legyen. Ebből adódóan csak az elosztott számítási rendszer üzemeltetői és a projektjeik vannak az adatbázisban tárolva. A részletes információk láthatóak az 5.1 ábrán.



5.1. ábra. Adatbázis szerkezetét ábrázoló diagram

Mint az a diagramból látható, a felhasználó és a projektek között egy az egyhez viszony van. Ez inkább követelmény szerinti megkötés, mint technikai, megelőzve ezzel azt, hogy túl sok párhuzamos projekt leterhelje az alkalmazást. Ezért felhasználónként egyszerre csak egy projekt megengedett. Persze az adott projekthez tartozó adatok és a projekt kódja bármikor változtatható.

Az adatbázisban található táblák szerver oldalon Java bean-eknek felelnek meg. Az egyedüli különbség, hogy a bean-ek a **play.db.ebean.Model** osztályból származnak és fel vannak annotálva JPA specifikus módon, hogy az Ebean ORM használható legyen. Tekintsük a következő példát:

```
@Entity
public class User extends Model
        @Id
        private String username;
        private String email;
        private String password;
```

Amint az látható, csak a user táblában szereplő oszlopok szerepelnek, mint adattagok, ahol a *username* mező az elsődleges kulcs és az osztályuk egy entitás. Ezek mellet csak a megfelelő getter és setter metódusok és egy konstruktor szerepel, mely tartalmazza az összes adattagot.

A password mező itt már egyszerű String, de amint beérkezik a szerverhez el van hash-elve biztonsági okokból, ennek részleteiről majd a későbbiekben olvashatunk.

5.1.2. Az MQ implementációja

Eredetileg egy saját implementációt hoztunk létre mely a Java Collections Frameworkre épített. Az elvont gyár tervezési mintát és a Singletont használta együttesen.

Két fő osztályból állt a megoldás, név szerint a Store és a Data osztályokból. A Store egy sigleton (egyke) várakozási sor, mely bármilyen Data típusú adatokat tárol. A Data meg egy generikus típusú osztály bizonyos segédfüggőségekkel. Az 5.2 ábrán a megvalósítások egy rövid kivonata látható.

```java
public class Data<Template> {
    private Template[] data;
    private int id;

    public Data(Template[] data) { ...

    }

    public Template[] getData() {
        return data;
    }

    public Integer getId() {
        return id;
    }

    // TODO: Add Template type
    public static Double[] getTemplateArray(String[] source) { ...
    }

    private void log() {
        System.out.println(this.toString());
    }

    @Override
    public String toString() { ...
    }

}
```

```java
public class Store<T> {
    private Queue<Data<T>> queue;
    private ConcurrentHashMap<Integer, Data<T>> inProgress;
    private List<T> results;
    private static Store<Double> doubleInstance = null;
    private static Store<Integer> integerInstance = null;
    private static Store<Float> floatInstance = null;

    private Store() { ...
    }

    public void addToStore(Data<T> elment) {
        queue.offer(elment);
    }

    public Data<T> getNextTask() { ...
    }

    public void taskFinishedSuccessfully(Integer id, T result) { ...
    }

    public List<T> getResults() {
        return results;
    }

    private static <T> Store<T> getInstance() {
        return new Store<T>();
    }

    public static Store<Double> getDoubleInstance() { ...
    }

    public static Store<Integer> getIntegerInstance() { ...
```

5.2. ábra. Saját Queue megvalósítás

Azonban azt a szintű tranzakcionalitást és perzisztenciát elérni, amit más MQ rendszerek adnak alapértelmezetten, nem lett volna kifizetődő implementálni. Ezért ezt lecseréltem egy kész termékre, melynek teljesítménye és szétoszthatósága az adott körülményeknek megfelel.

Az Apache ActiveMQ önmagában egy különálló alkalmazás és hozzá a szerver a JMS API-t felhasználva csatlakozik egy egyszerű alapértelmezett üzenetkezelő, egy úgynevezett brokeren keresztül. Minden projekthez egy várakozási sor tartozik, melyek neve a projekt tulajdonosával megegyezik. A felhasználó által feltöltött CSV fájl soronként külön üzenetekbe csomagolva a megfelelő sorhoz lesz hozzáfűzve.

Egy kérés bejövetelekor a sorban következő elem kikerül és továbbítva lesz a megfelelő kliensnek.

5.1.3. A Controllerek implementációja

Az adatbázis és MQ kapcsolat fenntartása mellet a szerver a kliensek kiszolgálásával is foglalkozik. A kliensek kiszolgálása Play framework-ön belül kontrollerek segítségével történik. Minden kérés az elérési útvonal alap-

ján egy kontroller metódusához van hozzárendelve. Ez a leképezés a *youshare/conf/routes* fájlban található meg és következőképpen néz ki:

```
POST    /api/v1/verify   controllers.UserController.isUserValid()
POST    /api/v1/upload   controllers.FileUploadController.upload()
POST    /api/v1/project  controllers.GridProjectController.submitGridProject()
GET     /api/v1/project/code/:gridProjectUserId
 controllers.GridProjectController.getGridProjectCode(gridProjectUserId)
GET     /api/v1/project/data/:gridProjectUserId
 controllers.GridProjectController.getGridProjectDataFromQueue(gridProjectUserId)
GET     /api/v1/project/:gridProjectUserId
 controllers.GridProjectController.getGridProject(gridProjectUserId)
GET     /*file           controllers.Assets.at(path="/public/app", file)
```

Amint az látható, minden leképezés 3 részből áll: a HTTP metódus, az elérési útvonal, és a metódus elérési módja. A kontrollerek a *play.mvc.Controller* osztályból származnak, ez biztosít bizonyos közös funkcionalitást és a keretrendszer működését.

UserController

Amikor az elosztott számítási rendszer tulajdonosa vagy más általa felhatalmazott felhasználó el szeretné érni a nem publikus oldalakat, akkor be kell jelentkeznie. Ilyenkor a felhasználó által megadott adatok a UserController-nek vannak továbbítva, mely főként azért felelős, hogy belepéskor autentikációt hajtson végre a felhasználókon.

Amennyiben az autentikáció végeredménye sikeres, a válasz HTTP 200 kóddal és a "valid" karakterlánccal tér vissza, különben a HTTP 401 kódot téríti vissza.

FileUploadController

Amikor egy projekthez fel szeretné tölteni a felhasználó a számítási adatokat tartalmazó CSV fájlt, akkor a kliens oldal a FileUploadController-hez egy HTTP POST-on keresztül tölti fel. A kész fájlt a kontroller átnevezi a felhasználó nevére, majd egy háttérszál segítségével feldarabolja és betölti a megfelelő várakozási sorba az ActiveMQ-n belül.

GridProjectController

A GridProjectController felelős az összes projekttel kapcsolatos szolgáltatások kezeléséért. Amikor egy felhasználó csatlakozik az elosztott számítási rendszerbe és a konkrét végrehajtandó Javascript kódot kéri le, a getGridProjectCode metódus hajtódik végre. Ez a a metódus a paraméterként kapott felhasználó név alapján lekérdezi az adatbázist és visszatéríti a megfelelő programkódot.

Ha a számítási probléma adatszeletet is igényel, akkor a getGridProjectDataFromQueue metódus csatlakozik az MQ-hoz, kiveszi a következő üzenetet, mely valójában egy adatszelet és válaszol a kliensnek. Amennyiben elfogytak az üzenetek, egy "[done]" karátérláncot küld. Ez jelzi a kliensnek, hogy kész a számítási feladat.

Abban az esetben, ha egy felhatalmazott felhasználó bejelentkezik és módosítani szeretné meglévő projektjét, a getGridProject az adatbázisból kiolvassa és visszaküldi a projekt összes attribútumát. Amennyiben nem létezik a projekt, a kliens oldalnak egy HTTP 404-el válaszol.

5.2. A kliens implementációja

A youshare alkalmazás keretein belül van egy webes kliens, mely lehetővé teszi, hogy a felhatalmazott felhasználók számítási feladatot definiáljanak, valamint az önkéntesek számára egy felületet biztosít, amely a felajánlott számítási kapacitásuk kihasználását biztosítja.

5.2.1. Az app.js implementációja

A kliens oldali kód legfőbb belépési pontja az *app.js*, amely létrehozza az AngularJS futásához szükséges kontextust és definiálja a modulokat. Amint láthatjuk a következőkben az alkalmazás önmagában egy *youshareApp* elnevezésű modul. Ezenkívül használva vannak még *ngCookies, ngResource, ngSanitize, ngRoute*, melyek AngularJS beépített modulok. A felsoroltak felelősek a böngésző sütik kezeléséért, az alkalmazás irányításáért, a kapcsolattartásért, valamint a bejövő adatok védelméért az XSS (Cross-site scripting) ellen.

```
angular
  .module('youshareApp', [
    'ngCookies',
    'ngResource',
    'ngSanitize',
    'ngRoute',
    'auth',
    'ui.ace',
    'angularFileUpload',
    'ui.bootstrap'
  ])
```

További modulok rövid összefoglalója:

- **auth** egy saját kliens oldali authentikációs modul, melyről részletsebben olvashatunk a service-knél
- **ui.ace** a technológiáknál említett ACE szövegszerkesztő

- **angularFileUpload** fájlfeltöltést biztosító modul
- **ui.bootstrap** a bootstrap megjelenítési keretrendszer AngularJS változata

Mivel az AngularJS kliens egy *single page application*, a dinamikus oldalakat elérési útvonalakhoz kellett kapcsolni. Ez nézet útvonal-leképezés is, mely az *app.js*-ben van definiálva a $routeprovider-en keresztül. Egy ilyen leképezés 4 elemből áll: az elérési URL, a nézett sablon elérési útvonala, a nézettel kapcsolatos kontroller, és az adott nézett publikus vagy privát-e. Például a bejelentkezési oldal leképezése a következő:

```
$routeProvider.when('/login', {
       templateUrl: 'views/login.html',
       controller: 'LoginCtrl',
       requireLogin:false
     })
```

5.2.2. A servicek implementációja

AngularJS-en belül minden service sigleton, így ideálisak a közös szolgáltatások és adatok tárolására. A különböző modulok közti megosztást a AngularJS beépített függőségkezelő rendszere megoldja.

AuthService

Ez a service felelős a kliens oldali autentikációért. Két metódusból áll a *getUserAuthenticated*, mely eldönti a *youshareUser* böngésző süti alapján, hogy a felhasználó be van-e jelentkezve vagy sem, a másik a*setUserAuthenticated*, amely a paraméterként beérkezett felhasználó és boolean érték alapján létrehozza a sütit vagy törli azt. Ez a service a fentebb említett *auth* modul része.

LoginService

A LoginService felelős a felhatalmazott felhasználok bejelentkezésének kezeléséért. Amikor a felhasználó megadja a felhasználói nevét és jelszavát, akkor az *attemptLogin* metódus hívódik meg. Ez HTTP kapcsolatot nyit a szerverhez és leellenőrzi, hogy a megadott adatok hitelesek-e. Amennyiben hitelesek, meghívja a fent említett AuthService.setUserAuthenticated metódusát a süti létrehozása érdekében.

ProjectService

A számítási feladatnak a szerkesztéséért és definiálásáért felelős oldalakról a szerverkapcsolatot igénylő műveletek vannak összegyűjtve a ProjectServicebe. Az ProjectService *uploadProject* metódusa HTTP POST-on keresztül feltölti a számítási feladattal/projekttel kapcsolatos információkat. Ennek a

fordított metódusa a *getProjectForUser*, mely a szervertől az adott felhasználó által definiált projektet kéri le.

A ProjectService-t felhasználó helyeken általbán szükség van a felhasználónévre is, ezért található még egy *getUserNameFromCoockieStore* segédmetódus is, amely a süti alapján visszatéríti a bejelentkezett felhasználó nevét.

5.2.3. A kontrollerek és nézetek implementációja

Ahogyan az már a Felhasznált technológiák, eszközök és módszerek részben is említve volt, az AngularJS keretrendszeren belül a kontrollerek fő szerepe a nézetek és az adatok irányítása. Ezért minden nézethez tartozik egy kontrollert definiáló fájl. A nevük konvenció alapján azonos, csak a fájl kiterjesztése más. Például *project.js* tartalmazza a *ProjectCtrl*-t és a *project.html* pedig a nézetet. Mivel ez a két fogalom szorosan összekapcsolódik, ezért a továbbiakban együtt fogom bemutatni a kontrollereket és a hozzá tartozó nézeteket.

ProjectCtrl

Ha egy felhatalmazott felhasználó rálép a projekt definíciós oldalra, akkor a ProjectCtrl lekérdezi a szervertől az adott felhasználó projektjét és betölti a projekt modellbe. Amennyiben nem létezik, akkor egy sablonprojektet tölt be. A modellt ezek után továbbítja a nézetnek, ahol megjelenik az, és szerkeszthető lesz a project neve és leírása.

Amint a felhasználó befejezte a szerkesztést és rákattint a tovább gombra, a *next* metódus frissíti az adatmodellt a nézetben létrejött változtatásokkal, ezután feltölti a szerverre, majd továbblép a következő oldalra.

EditorCtrl

Az előzőekben leírt lépések alapján láthatjuk, hogy a felhasználó a számítási feladatott definiáló oldalon találja magát. Ez a nézet egy ACE szövegszerkéztőt tartalmaz. Az ACE szövegszerkesztő a nézet betöltődésekor *aceLoaded* metódus segítségével lekéri az adatmodelltől a kódot és beletölti a szövegszerkesztőbe az egyszerűbb szerkeszthetőség érdekében.

Hasonló módon a *next* metódus frissíti az adatmodellt a szerkesztőben létrejött változtatásokkal. Mindezt feltölti a szerverre, majd továbblép a következő oldalra.

DataCtrl

Amennyiben a fent definiált feladathoz adatszeletet is szeretne csatolni a felhasználó, akkor mindezt megteheti a data nézeten keresztül egy CSV fájlt felhasználva. Amennyiben a felhasználó HTML5-ot támogató böngészőkben látogatott az oldalra, már drag-and-drop módon is működik a kijelölt területen belül, különben pedig használható a megszokott form feltöltés gomb. Mindez a *DataCtrl* segítségével van megvalósítva az *onFileSelect* metóduson

keresztül, mely a csatolt fájt feltölti a szerverre és közben frissíti a nézetet a folyamat állapotának megfelelően. A *next* metódus itt átirányítja a nézetet a Done oldalra.

DoneCtrl

A done oldalon a felhasználó megtalálja azt az URL-t, melyet meg tud osztani az önkéntesekkel, hogy becsatlakozzanak a felhasználó elosztott számítási rendszerébe. Ennek az URL-nek az előállításáért és a nézettel való megosztásáért felelős a DoneCtrl.

MainCtrl

A főoldalon levő elemeknek segít a MainCtrl eldönteni, hogy megjelenhetnek vagy nem mindezek, és ezt az alapján teszi, hogy az adott felhasználó be van-e jelentkezve vagy sem.

ShareCtrl

Abban az esetben, mikor egy önkéntes rálép a számítási oldalra, amely segítségével becsatlakozik a számítási rendszerbe, a ShareCtrl a következőkért felelős: a projekt letöltése, modell létrehozása, majd átadása a nézetnek, hogy megjelenítse a címet és leírást, a projekt programkódjának a hozzáfűzése a nézethez, hogy a böngésző elvégezze a számításokat.

6. fejezet

Mérési adatok

A Youshare projekt fő célja többek között egy hasonló rendszer Javascript-es változatának a használhatóságának és teljesítményének a vizsgálata. Ebben fejezetben a létrehozott rendszernek teljesítményével kapcsolatos méréseket és az Apache JMeter-t, mint felhasznált mérési eszközt mutatjuk be.

6.1. Apache JMeter

Az Apache JMeter egy teljesen nyílt forráskódú Java-ban írt program. Ez segít terhelést és látogatottságot szimulálni egy szerverre, miközben statisztikákat gyűjt. A következő protokollokon keresztül tud csatlakozni a mért rendszerekhez:

- Web - HTTP, HTTPS
- SOAP
- FTP
- Database via JDBC
- LDAP
- TCP

6.2. A mérések részletezése

A méréseket három fő kategóriára bontottuk, ezek lefedik a teljes rendszert. Ilyen a HTML, mely csak egyszerűen a fájl lekéréseket nézi a publikus

mappából, a DB, ez olyan lekérdezést végez, ami az adatbázisból igényelt adatokat használja fel, és az MQ, amely az ActiveMQ-ból kér le adatokat.

A tesztkörnyezet 2 számítógépből állt. Az egyik volt a szerver, a másik pedig a klienseket szimulálta JMetert felhasználva.

A teszteket megismételtük 10, 100 és 1000 felhasználót szimulálva. Mindezt a lehető legnagyobb ismétlések számával, a lehető legnagyobb mintakészlet elérése érdekében. Próbálkoztunk 3000 felhasználó szimulálásával is, de a tesztszámítógép kapacitása nem volt elegendő, így csak hiányos adatokat sikerült összegyűjtenünk.

A JMeter által készített adatokat egy CSV fájlba mentettük, ahol csak az egyes lekérések teljes válaszideje, a HTTP státusza, a sikeres adat mérete, és a latency szerepel. A latency a kérés megkezdése és a válasz beérkezése közötti időintervallum. Ezek segítségével kiszámoltuk az átlagot és a szórást, hogy megtudjuk határozni a konfidencia-intervallumot. A számításokhoz a következő képleteket használtuk:

Az átlaghoz $\mu = 1/n * \sum_{i=1}^{n} x_i$

Az szóráshoz $\sigma = \sqrt{1/n * \sum_{i=1}^{n} (x_i - \mu)^2}$

A konfidencia-intervallum (95%-os valószínűséggel) $CI = \mu \pm \frac{1.96 * \sigma}{\sqrt{n}}$

Ezek alapján a fájl lekérések idejei a következők voltak. A mérési idők milliszekundumban vannak meghatározva.

Nr. users	Min	Max	Median	Average	Std Deviation	Confidence Interval
10	13	164	49	52,1	23,86	[50,62 ; 52,22]
100	22	783	603	584,74	92,73	[582,92 ; 614,46]
1000	18	20647	6859	9454,08	5218,81	[9421,18 ; 9486,97]
3000	18	20647		12076		

Amint látjuk, 100 felhasználó esetén is megközelítőleg 0.5 másodperc a teljes válaszidő, ami nagyon jó teljesítménynek számít. De az 1000 felhasználó válaszideje se több 9.5 másodpercnél. A továbbiakban áttekintjük az adatbázist érintő lekérdezések eredményeit.

Nr. users	Min	Max	Median	Average	Std Deviation	Confidence Interval
10	8	92	17	19,97	10,64	[19,31 ; 20,62]
100	11	367	276	272,49	32,66	[271,84 ; 273,13]
1000	11	4279	2687	2300,97	777,6	[2295,58 ; 2306,35]
3000	11	3369		1754		

Összehasonlítva ezt a HTML fájl szerveres eredményekkel szignifikáns csökkenés látható. Feltehetőleg a merevlemezről való olvasási idő hiányát láthatjuk a megközelítőleg 7 másodperces csökkenéssel az 1000 felhasználó esetén.

Végül nézzük meg az MQ-t érintő mérési eredményeket. A többiekhez képest ez a legszignifikánsabb. Amíg az első két eset felhasználónként feltehetőleg egyszer fordul elő, az oldalra első látogatáskor az MQ-t érintő lekérések folyamatosan érkeznek a részfeladatok elvégzése esetén.

Nr. users	Min	Max	Median	Average	Std Deviation	Confidence Interval
10	25	2126	116	120,39	74,15	[115,79 ; 124,98]
100	23	11562	1939,5	2316,28	1706,37	[2255,21 ; 2377,34]
1000	60	236112	59236	79720,79	62235,05	[78414,34 ; 81027,23]
3000	200	236112		109115		

A fentiekből látható, hogy 1000 felhasználó esetén a válaszidő megközelítőleg 80 másodperc a legrosszabb esetben is. Bár ez jelentős késleltetést visz be a rendszerbe, de mivel részfeladatonként csak egyszer fordul elő, ez az eset még elfogadható. A késleltetés nagyrészt valószínűleg az MQ-val való kommunikációból származik.

7. fejezet

Következtetések és további fejlesztési lehetőségek

A projekt fejlesztése során sikerült egy olyan általános célú elosztott számítási rendszert létrehozni, amely kevésbé függ a rendszer üzemeltetőjének hardverétől. Tiszta design segítségével nagyon jó teljesítményt sikerült elérni 100 becsatlakozott önkéntes esetén, de 1000 önkéntes esetén is elfogadható a rendszer teljesítménye.

Az AngularJS segítségével sikerült egy kellemes felhasználói felületet létrehozni, és emellett sikerült a terhelést átvinni szerver oldalról a kliensek felé.

Az ActiveMQ által nyújtott segítséggel sikerült megoldani a párhuzamos lekérések szálbiztonságossá tételét. Az ActiveMQ által nyújtotta lehetőségeket felhasználva akár más számítógépről is futtatható lenne, vagy több példányban is elindítható lenne, egy terhelés elosztóval. Ez a terhelés elosztó pedig mindig a legkevésbé foglalt MQ példányhoz irányítaná a forgalmat.

A projekt viszonylag könnyen testre szabható és bővíthető, mivel az interface-ek jól meghatározottak, de akár az ActiveMQ is könnyen lecserélhető egy másik JMS implementációra.

A projekt az 1000-es nagyságrendű önkéntes méretű projektek számára ideális. A további növekedéshez fejlesztést igényel.

Hosszútávú fejlesztési ötletünk, hogy nyílt forráskódúvá tennénk a Youshare project-et, ezáltal is közelebb vinni az átlagfelhasználókhoz a elosztott számítási rendszerek teljesítményét. Továbbá az MQ-n keresztüli válaszidő csökkentése érdekében kísérletezést kellene végrehajtani azzal, hogy egy feltöltött fájlt több várakozási sorba osszuk szét, ezáltal is csökkentve egy-egy sor terhelését.

Ezenkívül, valós felhasználás esetén mindenképp szükséges lenne SSL használata, hogy a kérések HTTPS-en keresztül menjenek, ezzel is védve

a felhasználói adatok biztonságát.

Egy harmadik továbbfejlesztési lehetőségként érdemes lenne egy cacheelés kialakítása az ismételt adatbázis kérések gyorsítása érdekében. Természetesen az alkalmazás hosszútávú használata során derül majd ki csak igazán, hogy milyen további fejlesztések lesznek elengedhetetlenül szükségesek a jövőben.

Irodalomjegyzék

[1] Alex Rodriguez,RESTful Web services: The basics http://www.ibm.com/developerworks/webservices/library/ws-restful/

[2] Christophe Cérin,Gilles Fedak, Desktop Grid Computing CRC PRESS 2012.

[3] Douglas Crockford, JavaScript:The Good Parts O'Reilly Media, Inc. 2008.

[4] Iványi Antal, Angol-magyar informatikai szótár http://www.tankonyvtar.hu/hu/tartalom/tkt/angol-magyar/adatok.html

[5] Joshua Bloch, Hatékony Java Kiskapu Kft. 2008.

[6] Mark Hapner, Java Message Service http://docs.oracle.com/cd/E19957-01/816-5904-10/816-5904-10.pdf

[7] Martin Fowler,Language Workbenches: The Killer-App for Domain Specific Languages? http://www.martinfowler.com/articles/languageWorkbench.html

[8] Mironx Sadziak,Simple guide to Java Message Service (JMS) using ActiveMQ http://www.javablogging.com/simple-guide-to-java-message-service-jms-using-activemq/

[9] Nicolas Vahlas, Some thoughts on stress testing web applications with JMeter http://nico.vahlas.eu/2010/03/30/some-thoughts-on-stress-testing-web-applications-with-jmeter-part-2/

[10] Shekhar Gulati, Day 30: Play Framework–A Java Developer Dream Framework https://www.openshift.com/blogs/day-30-play-framework-a-java-developer-dream-framework

[11] Typesafe Inc.,Play framework hivatalos oldala http://www.playframework.com

[12] Typesafe Inc.,Scala hivatalos oldala www.scala-lang.org

[13] Typesafe Inc.,Akka keretrendszer hivatalos oldala http://akka.io

[14] Y.Shafranovich,Common Format and MIME Type for Comma-Separated Values (CSV) Files http://tools.ietf.org/html/rfc4180

[15] ***, Apache ActiveMQ hivatalos oldala http://activemq.apache.org

[16] ***, Bower git repository hivatalos oldala https://github.com/bower/bower

[17] ***, Grunt hivatalos oldala http://gruntjs.com/

[18] ***, H2 hivatalos oldala http://www.h2database.com/html/main.html

[19] ***, H2 hivatalos oldala http://www.h2database.com/html/main.html

[20] ***, JSON hivatalos oldala http://json.org/

[21] ***, UI ACE hivatalos oldala http://angular-ui.github.io/ui-ace/

[22] ***, Yeoman hivatalos oldala http://yeoman.io/