UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science
Information Technology

Rein Torm

Cloud-based data analysis and server scalability for SportID

Bachelor's Thesis (6 ECTP)

Supervisor(s): Satish Narayana Srirama
Pelle Jakovits

Tartu 2014

Cloud-based data analysis and server scalability for SportID

Abstract:

This thesis explores two issues that concern SportID’s web application. The first problem is linked to the current server architecture. The author is going to find out if it will be enough in case of a bigger load and how it can be scaled further using load balancing technique. The second problem is the lack of an affordable data analysis tool that suits SportID. The author is going to create a simple cloud based data analysis solution and tries to provide the rationale for making this solution.

Keywords:

Load balancing, auto scaling, cloud-based data analysis, Hadoop

Pilvetehnoloogial põhinev andmetöötlus ja serveri skaleeritavus SportID rakendusele

Lühikokkuvõte:

See lõputöö uurib kahte probleemi, mis puudutavad SportID veebirakendust. Esimene probleem on seotud serveri arhitektuuriga. Autor uurib, kas praegune serveri arhitektuur on sobiv ka suurema koormuse puhul. Teiseks probleemiks on SportID vajadustega sobiva odava andmeanalüüsi süsteemi puudumine. Autor loob lihtsa andmeanalüüsi platvormi kasutades selleks erinevad pilvetehnoloogia andmetöötlussüsteeme ja võrdleb loodud süsteemi tavalise relatsioonilise andmebaasi analüüsimisega.

Võtmesõnad:

Serverite koormuse jaotus, automaatne serverite skaleeritavus, pilvetehnoloogial põhinev andmeanalüüs, Hadoop

Table of Contents

1 Introduction

Internet has become the main resource of information for many people. With the last decade many web applications have become part of peoples' everyday life. With the fast growth of web applications many issues may emerge for the application creators.

First issue is linked to the usage of web applications. When a web application gets more popular it naturally needs more resources to remain fast and provide a pleasant experience for the customer. Arguably the most famous social network Facebook had 135 million new users in 2011 [1]. According to calculations made in the article [2], in 2012 Facebook had about 180,000 servers. This makes 750 users per server. This alone proves that even starting web applications need to worry about hardware and where to get it. To this problem lies a possible solution in the cloud service providers like Amazon and Rackspace [3], who offer virtual servers which can be started on demand and in few a moments - allowing users to scale the number of servers up or down in real time and distribute the user requests between them.

The second issue is linked to the data that users produce. All the clicks, page visits, posts, etc. people create in the web application are usually saved as some form of data. Unfortunately data without someone understanding it is useless. This data can be a powerful tool to steer business into further success. According to article [4], Target, American retailing company can predict using its sales information when a woman is pregnant. Using this information they can make special sales offer for the pregnant customers. To analyse vast amount of data we need necessary tools. Fortunately big companies as the one mentioned above have already run into this problem and have developed open-source tools which any developer can get their hands on.

In the current thesis the author is going to study the load balancing and cloud-based data analysis on one Estonian start-up's web application. The goal of this work is to modify the current SportID's system to enable it to support far larger number of users in the future and also create a scalable data analysis tool that could be used and modified by SportID's programmers in the future.

1.1 SportID

SportID is a start-up that emerged from the Startup Wise Guys accelerator. SportID's web application currently runs on one server that is hosted using Rackspace cloud service.

Companies which compensate their employees sporting habbits are the main clientele of SportID. SportID offers them a platform to manage their employees sports compensations. SportID can also be used for registering onto different sport events like marathons. Users can use their sports compensations or transfer money on their account and use this money to pay for sporting events. Registering is easier because user information is already in SportID’s system. Thus it takes only few clicks to register to an event.

Since SportID is planning to open its registration to everyone and also plans to go to foreign markets, there may be a need for load balancing and tools to analyse larger amounts of data.

1.2 Web-application scalability on Rackspace

Every month there are multiple companies who start using SportID’s service. These companies bring 2-1000 new users who start using SportID’s web application. Due to the growth of the user-base, server needs to handle more load. To handle more load, there is a need for a bigger server. One server can only handle a certain amount of requests per second, since it cannot have infinite number of threads that handle the requests. To solve this problem, there is a possibility to „scale-out“. This means that rather than using one big server, it is possible to use multiple servers that share the load. In this thesis we are going to study how to do load-balancing on Rackspace cloud server and see if this kind of scaling is really beneficial to the user.

1.3 Cloud-based statistics analysis tool

Currently there is no statistics module that is efficient, cheap enough and knows how to turn database data into meaningful statistics for SportID. To solve the problem, the goal is to make a scalable cloud-based data analysis module that is going to analyse SportID’s gathered data. The statistics module is meant to handle the data created by a user base of 100 000 registered users.

2 State of the Art

2.1 Scaling web applications

The scaling of applications has changed drastically over the last decade. When we go back ten years, there was no cloud computing platforms available for the bigger public. There was no easy way to rent couple of virtual machines (VMs) and scale your application on those machines. According to Simon Bisson [6], large-scale web systems needed to have several high-powered servers, a bunch of storage and high-end networking devices. This all cost tens of thousands of dollars. This all started to change after year 2000. With Amazon in the lead, new possibilities started to emerge. Amazon, having found that cloud architecture was much more efficient, started offering this possibility to external customers in 2006, by launching Amazon Web Services(AWS) [7]. The pricing principle was and still is to pay only for what you use. Soon after Amazon other cloud computing vendors started to emerge. For example, in March 2006 Rackspace announced their cloud-computing service.

2.2 Benefits of Cloud computing

Scalability and auto scaling

The ability of the system to deploy extra resources when there is a growth of workload on the application is called scalability. This can result in an extra VM being deployed. However this does not show how well the extra resources are matched to the growth of workload. For example when the workload increases by 10 new users, new VM is deployed, however there was a need for only half of that power so we should use a smaller machine for the job.

Auto scaling enables to automatically launch and remove VM instances. The automatical launch or removal is defined by settings the owner has set to the whole system. There are multiple ways to use auto scaling.

The easier option is for the owner to define the time frames when extra load is expected. For example a business owner knows that from 5 p.m. to 6 p.m. there are 50% more visitors on the web page than usually. Owner then chooses to set auto scaling setting that fires up an additional server every day between 5 to 6 p.m.

The second option is used when there is no possible way to predict when the traffic spikes. In this case the owner can define the criterion which triggers the addition of new resources. For example, if web application reaches 1000 simultaneous visitors, new VM is deployed. In this case however there are few weak points. Since the deployment of the server may take couple of minutes, customers may not receive good user-experience immediately. Also if there are no limits set to scaling, upon a distributed denial-of-service (DDoS) attack [8] auto scale will assign more and more servers to the web application and owner will lose a lot of money.

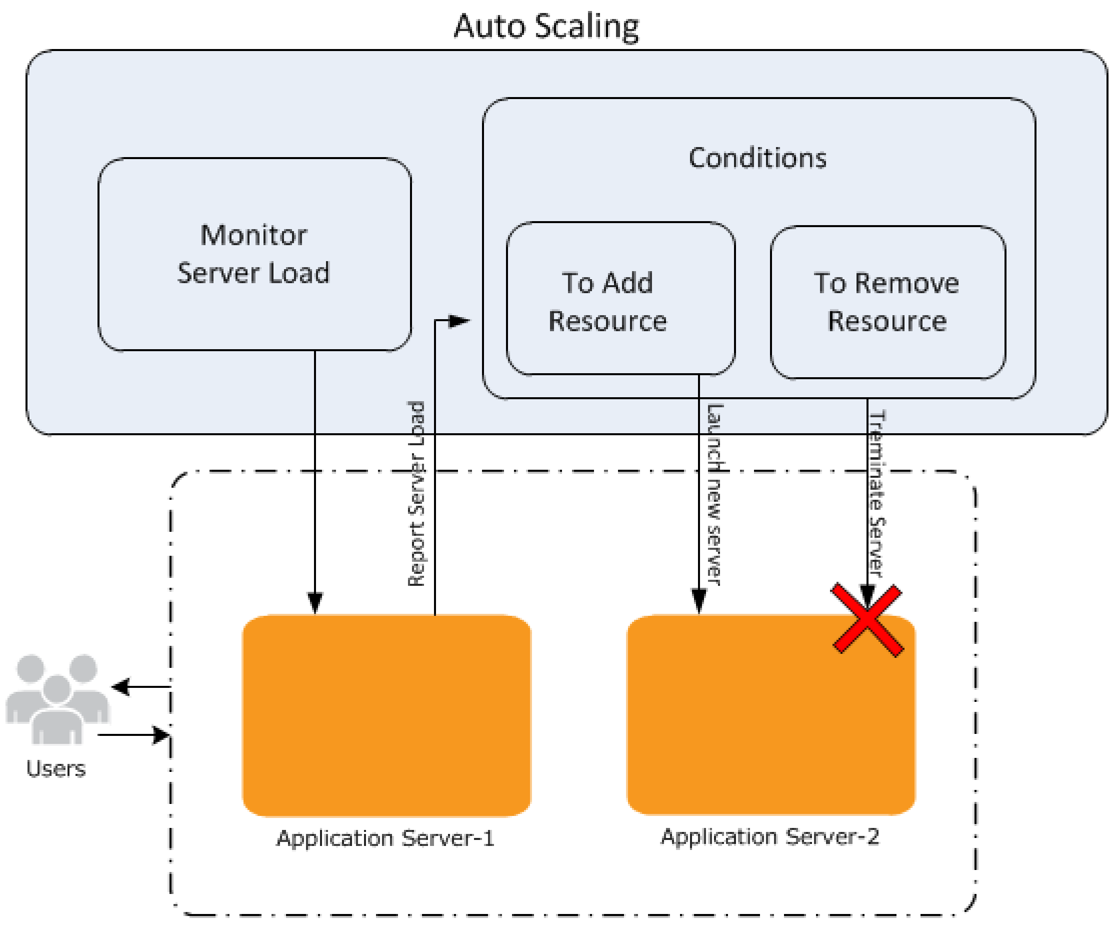


Figure 1. *Auto Scaling Architectural Diagram [19]*

Elasticity

The ability of the system to adapt with workload changes by provisioning and deprovisioning resources. The goal is to match the demand as close as possible.[9].

The aim of elasticity is to avoid over- and under-provisioning. Over-provisioning means that more resources are being allocated than there is the need for. Under-provisioning is the opposite; fewer resources are being allocated than there is the need for.

Elasticity's core aspects are **speed** and **precision** [9]. Speed defines how fast we can return to optimal state from over- or under-provisioned state. Precision on the other hand is determined by the offset of the actual need of resources.

The main concern with elasticity is that it takes time. Deploying a VM usually takes time. It can take up to several minutes. The boot start up time is dependent on many factors like the size of the VM, the location of the server park, etc.

Reliability

One of the big reasons to choose cloud hosting service over buying servers is the reliability. Cloud service providers like Rackspace and Amazon EC2 promise the clients uptime from 99.95% to 100%. Providers promise that even on the event of natural disaster service prevails.

This can be achieved by making back-ups from the clients' web application and duplicating them on different servers. For example when your web application is hosted on some Amazon server and that server fails; the duplicate of your web application will be started on the backup server. Your clients and you will probably never see the outage. Even if we want to be extra careful and protect our application against power outages and natural disasters that might happen near provider's server parks, there is a possibility to just run an extra instance on a different continent. Most cloud providers offer this option.

2.3 Amazon vs Rackspace

There are many different cloud service providers like Amazon EC2, Rackspace, Windows Azure, GoDaddy, Singlehop. The author is going to compare Amazon and Rackspace, since he is the most familiar with them and has hosted web applications on both service provider VMs.

Locations

Rackspace: Dallas(US), Chicago(US), Northern Virginia(US), London(EU), Hong Kong(Asia), Sydney(Australia)

Amazon: Oregon, Northern California, Northern Virginia, Sao Paulo, Ireland, Sydney, Tokyo, Singapore

As we can see, both providers have data centers in most continents. None of the two have a data center in Africa, Rackspace also does not have a data center in South America and on the east-coast of the US. Having a data center near the clientele, can be very important when web application needs to react very fast to the user's actions (e.g. shooting games).

Difference between instances

Provider	Instance type	vCPU	Memory(GiB)	Storage(GB)	Price (€/h)	Band-width (Mb/s)
Amazon	Small	1	1.7	160	0.032	Unknown
Rackspace	Small	1	1	20GB SSD	0.029	200
Amazon	Medium	1	3.75	4 SSD	0.051	Unknown
Rackspace	Medium	2	2	1 x 40 SSD System storage 1 x 20 SSD Data storage	0.058	400
Amazon	Large	4	15	2 x 40 SSD	0.2	500
RackSpace	Large	4	15	1 x 40 SSD System storage 1 x 150 SSD Data storage	0.49	1250
Amazon	XL	8	30	2 x 80 SSD	0.41	1000
Rackspace	XL	8	30	1 x 40 SSD System storage 1 x300 SSD Data storage	0.98	2500

Table 1 Small to medium sized instances comparison [10][11]

All the information used in Table 1 is taken considering that instances run on Linux based operating system. The instances shown in Table 1 are just a few from the total number of offered instances by both providers. These instances were chosen because they have similar properties and are relevant in the latter sections.

As we can see, the Small instances have the same amount of vCPUs (virtual central processing unit). The main difference is presented in the storage and memory. Rackspace instance offers faster storage, but has less memory.

Medium instances differ pretty much from each other. Amazon instance strength is its bigger memory size and slightly smaller price. Still, if we consider having a medium sized web application with a proper database and functionality, the Amazon storage size may be a little bit too small. One option would be to get an external storage, but this will cost extra and will probably require some extra work to set up.

Large and XL type instances have the same amount of CPUs and memory. The difference is in the storage size and bandwidth speed. Rackspace storage setup is different from Amazon's. Rackspace servers use two different storages. The point of system storage is to store the application and perhaps some important data. When a snapshot is made from an instance, it only takes the system storage data into snapshot. This means that server creation from existing snapshot will be faster but the data in data storage will not be included to that server automatically. For data storage, there is a need to do extra backup.

2.4 Cloud-based data-analytics

With the growth of internet and web applications more and more data is created. According to IBM we create 2.5 quintillion bytes of data every day [12]. Walmart, an American multinational retail corporation alone has 1 million customer transactions every hour. This data alone bares no meaning. To make useful statistics out of this data, regular database queries are not enough. They are not enough because for one, the data can be unstructured and two, queries on relational databases cannot handle this amount of data.

To analyse big data, we need proper tools. Big web applications use multiple servers for load balancing. To analyse big data we can also use multiple servers/computers also called nodes, to divide the workload. This way large chunks of data can be processed faster into meaningful results.

Apache Hadoop

There is only one widely known framework for big data analysis, called the Apache Hadoop. Apache Hadoop is an open-source framework that allows us to analyse large data sets on clusters of computers (nodes). Hadoop does not require expensive hardware for computation; this framework is designed to run on commodity hardware. This means that the framework is built in a way that it detects broken nodes and handles the failures without losing any information.

Hadoop MapReduce

Hadoop MapReduce is a software framework used to write applications that process the large sets of data in-parallel on large clusters. These clusters can consist of thousands of nodes. MapReduce job splits the data into different chunks that are then processed on nodes in-parallel. MapReduce framework views the input as key-value pairs. MapReduce jobs in Hadoop are written using Java programming language.

MapReduce can consist of 5 steps:

- Map – Master node takes the input and divides it into data chunks as key-value pairs. These chunks are then sent to nodes.
- Combine – This step is not mandatory, but can be helpful in some cases. Mini-reducers that are used as an optimization to reduce network traffic.
- Partition – Divides up key space for parallel reduce operations
- Shuffle and sort – Data is parallel-sorted in order to move it from the map node to the node in which it will be reduced.
- Reduce – Takes the given values and does a specified function with these values. For example reduce can iterate through values and sum them up by key.

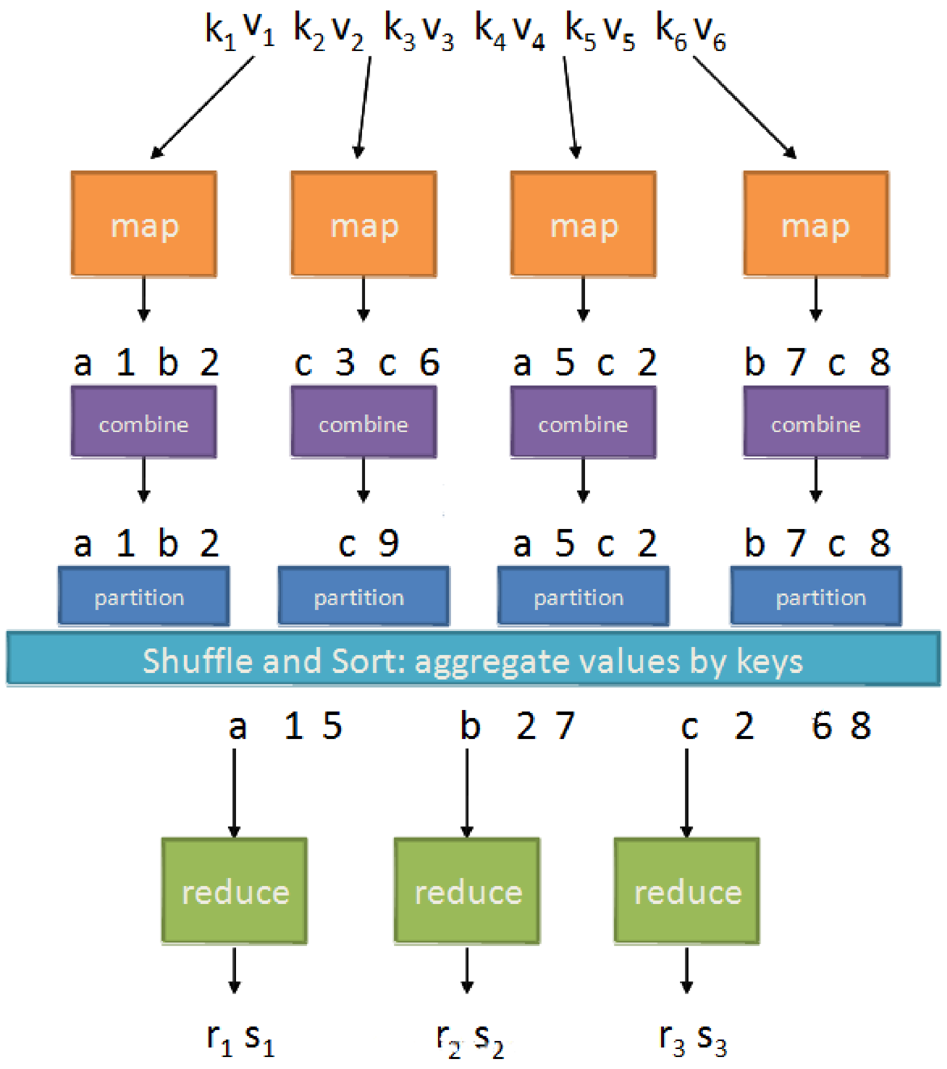


Figure 2. *MapReduce steps [20]*

Hadoop higher level query languages

Writing low level MapReduce code is slow and a lot of custom code is required for simple tasks. There is also a possibility to use high-level query languages to process large data sets. These languages are SQL-like (Structured Query Language) query languages which may help to speed up the process. Since most of the developers are familiar with SQL, it is a good way to process large data sets.

Apache Pig

Pig is a platform for creating MapReduce programs that can be used to analyse large data sets. Pig was originally developed by Yahoo. Pig has two different execution environments. First one is local mode where scripts are run on single machine. In the second mode, scripts are run in the Hadoop cluster. Pig Latin has some similarities to SQL but there are also many differences. Pig Latin statements are translated into MapReduce jobs which are passed to Hadoop for execution.

Apache Hive

Hive is a data warehousing built on top of Hadoop for data summarization, querying and analysis. It was originally developed by Facebook. Hive uses querying language called HiveQL which is very SQL like language with some differences. This means that the learning curve for developers who have previously used SQL is fairly small compared to Pig Latin or simple MapReduce. HiveQL statements are automatically translated into MapReduce jobs which are submitted to Hadoop for execution.

2.5 Conclusion

If we look back 10 years, web applications had to be hosted on a bought/rented server that did not have very many capabilities. We had to pay for unnecessary resources that were used only on occasion. With the current state of art, there is a possibility to only use and pay for the amount of resources our web application actually uses and also increase necessary resources as much as possible as soon as the need arises.

Large-scale data analysis has also become available for every developer and company who feels the need to gather vast amount of data and analyse it. Thanks to the open-source Hadoop framework and various platforms built on top of it, everyone with some programming skill can take a leap forward and try to analyse big data.

3 Architecture

3.1 Scaling on Rackspace

Introduction

SportID is constantly growing and there is a plan to expand the business to foreign markets, therefore the current server architecture needs to be rethought. In the following sections the author is going to introduce the current and the future server architecture and perform tests that measure the server performance and difference between two solutions.

Current solution

SportID’s web application is hosted on Rackspace virtual cloud. The web application is hosted on one Rackspace medium virtual machine instance mentioned in Table 1. The virtual machine uses Ubuntu [13] as its operating system. Server is situated in the Rackspace Chicago region.

The web application uses Apache HTTP server [14] as a server solution and MySQL relational database management system for data storage. Both, the web application and MySQL database are hosted on the same instance. This means that they share the resources allocated for the virtual machine. Web application itself is written on WordPress platform using the following programming languages: PHP, JavaScript, HTML and CSS.

Since in the current solution, both the web server and database use the same resources to work, these resources will run out more quickly than they would if the database was located on a different server.

New solution description

As mentioned in the previous section, separating the database and web application to different servers would open up more resources for both technologies. There is another benefit to this. If the database is hosted on separate server, it is possible to add more web application servers that can serve the users. These web servers will also connect to the existing database server to get and insert data. If each web server had its own database, the displayed data could be different each time the user visits the web application and this is not acceptable in any way.

When building the new solution, the author took into consideration that it should support the user base of 100 000 registered users. Currently there are 6072 registered users.

The new solution will use the same software as the current solution. This means that the database, server and operating system software will remain the same; only the amount of servers will change.

The new solution consists of:

- 1 load balancer
- 2 web servers
- 1 database server

Since there are two web application servers, there is a need to control the traffic flow between those two servers. That is where the load balancer becomes necessary. Load balancer's task is to direct the connecting clients to one of the web servers using a certain algorithm.

Creation of the new solution

First of all there is a need for new servers. The Rackspace environment allows the creation of an image from existing server instance. This image is basically a copy of the current state of the server. The image can be used to create new server instances. Since these instances are identical, they have the same web application and MySQL database installed on them.

As the first step, the author creates one new server from the existing image. Aside from same configuration and data, the new server also has the same amount of resources. On this server will be the database, also shown on Figure 3 as SQL1. Before this, the database was in the same server as the web application and did not need any extra configuration to work. Now, that it is on separate server, the MySQL database needs to be configured in order to accept connection from remote environments. The configuration includes two steps:

1. Log into MySQL server and open the MySQL default connection port 3306 with the following command:

```
sudo /sbin/iptables -A INPUT -i eth0 -p tcp --destination-port 3306 -
j ACCEPT
```

2. Log into MySQL to allow connections from external IP addresses. The following command will give all rights to external connections that connect to the MySQL server using the right user credentials.

```
GRANT ALL PRIVILEGES ON *.* TO 'username'@'%' IDENTIFIED BY 'password'
WITH GRANT OPTION;
```

3. As the last part of the configuration, the apache HTTP server needs to be shut down as it is not needed on the database server and is just using resources. Server can be shut down with a simple command:

```
sudo service apache2 stop
```

Now that the database is ready for usage, web application servers can be connected to database server to access the data. Since there already was one application server, shown as S1 on Figure 3, the author has to create only one new web application server. The new server is also created from the existing image, shown as S2 on Figure 3.

Previously both of the web application servers were connected to the local database, now they need to be configured to access data on the remote database server. Configuration is same for both servers:

1. In the web application root directory there is a file called *wp-config.php*. In the file line where SQL host is defined, the new database server IP address has to be assigned.
2. Since the web application servers do not require local MySQL servers anymore, for extra resources, the MySQL services need to be shut down. This can be done with command:

```
sudo service mysql stop
```

Now that there are two web application servers, there is a need for a load balancer. If there would be no load balancer, it would be impossible to share the load between two servers. Both of these servers would have a different IP address and therefore could not be accessed using the same address.

To create a load balancer, the user has to go through a few steps. User has to specify the region that the load balancer will be created to, assign the preferred load balancing algorithm to it and assign the two created web application servers as its nodes. Since both of the web application servers are identical when it comes to the resources, the author decided to choose „Least connections“ algorithm. This algorithm directs the connected user to the web server that is serving the least clients. The load balancer is shown on Figure 3 as LB1.

As the last part of the new solution, the Domain Name System (DNS) record needs to be set for the load balancer. Without this, users have to connect using an IP address and this is not convenient for users. Again Rackspace environment has a simple tab for editing DNS records. Previously the domain name was assigned to the initial server. This DNS record needs to be copied to the existing load balancer and deleted from the initial server.

Now that the configuration is finished, the load is balanced between two web application servers which get and save data to external database server.

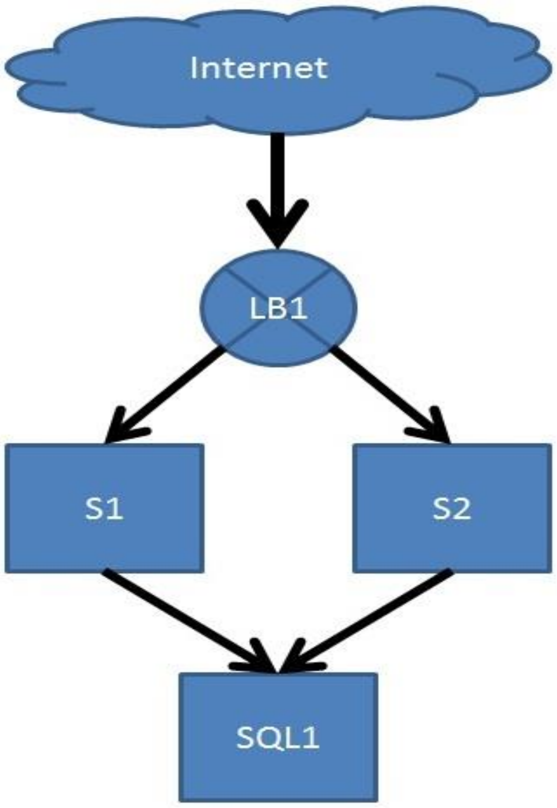


Figure 3. *SportID's load balancing solution.*

Test preparation

As mentioned in the section 3.2.3, the new solution should be able to support the web application with 100 000 registered users. This does not mean that there are 100 000 concurrent users. To perform the tests on the old and the new solution, one should know the number of concurrent users on the site. This can be calculated with a simple formula:

*Concurrent users = (hourly_visits) * session time(seconds) / 3600*

Since SportID uses Google Analytics (GA) [15] , the input data for the formula can be taken from there. The data taken from GA is the result of measurements in the timeframe of April 2,2014-May 2,2014.

Case	Hourly visits	Session duration(seconds)	Concurrent users
Maximum	163	145	6.6
Average	18.3	171.6	0.87

Table 2 *Site visitation data with current user base(6072 registered users)*

GA provides only the maximum value of hourly visits, but does not provide the average. Average hourly visits is calculated with formula:

Average hourly visits = Daily visits / 15

Daily visits are divided by 15 because the site is visited between 07:00 – 22:00(UTC +02:00).

Data in Table 2 is measured with the user base of 6072 registered users. To predict the amount of concurrent users when there are X amount registered users, following formula is used:

*Concurrent users(With X registered users) = X * concurrent user(6072 registered users)/6072*

Same can be done with hourly visits, only need to replace the concurrent user field in the formula with hourly visits field. The results with 100 000 concurrent users are show in Table 3.

Case	Hourly visits	Session duration(seconds)	Concurrent users

Maximum	2635	145	106
Average	301	171.6	15

Table 3 *Site visitation data with expected user base(100000 registered users)*

Test cases

As the current solution has supported the load for the current user base, there is no need to run tests for the Table 1 scenario.

Two tests will be performed on both server architecture solutions. The first test will use the average amount of concurrent users and the second test will use the maximum amount of concurrent users considering that there would be 100 000 registered users (Table 2 scenario). For testing the author decided to use Load Impact [16] testing tool. Load Impact opens provides the option to create test cases that consider multiple user scenarios. Each scenario can have a certain percentage of the concurrent virtual users executing it.

In both test cases two statistics are measured:

1. User load time (aggregated) – Sums the time that the simulated client took to perform all the HTTP transactions in the user scenario. Considering there were 3 scenarios, this time can vary a lot. Simple front page visit is one HTTP transaction, while logging in and looking at statistics is a longer process.
2. Clients' active – Shows how many concurrent users are using the web application.

Test case 1

As can be seen from Table 1 and Table 2, the session durations are different between the average and the maximum scale. This indicates that the user visit scenarios are different. When looking closer at GA statistics, it shows that on average, page visitations (%) were the following:

- 30% of the users were visiting blog
- 24% of the users the front page of the application
- 46% of the users where logging in and doing some actions like looking at their transaction history.

This means that there are 3 user scenarios that need to be considered. The first two user scenarios need to load mostly HTML, CSS, PHP and very few MySQL queries, meaning that the main resources consumption will occur in the web application server(s). The last user scenario includes different MySQL queries because the users are using the functions of the web application as they are logged in and this means retrieving data from the database. Therefore this user scenario will put a bigger load onto the database server than the first two.

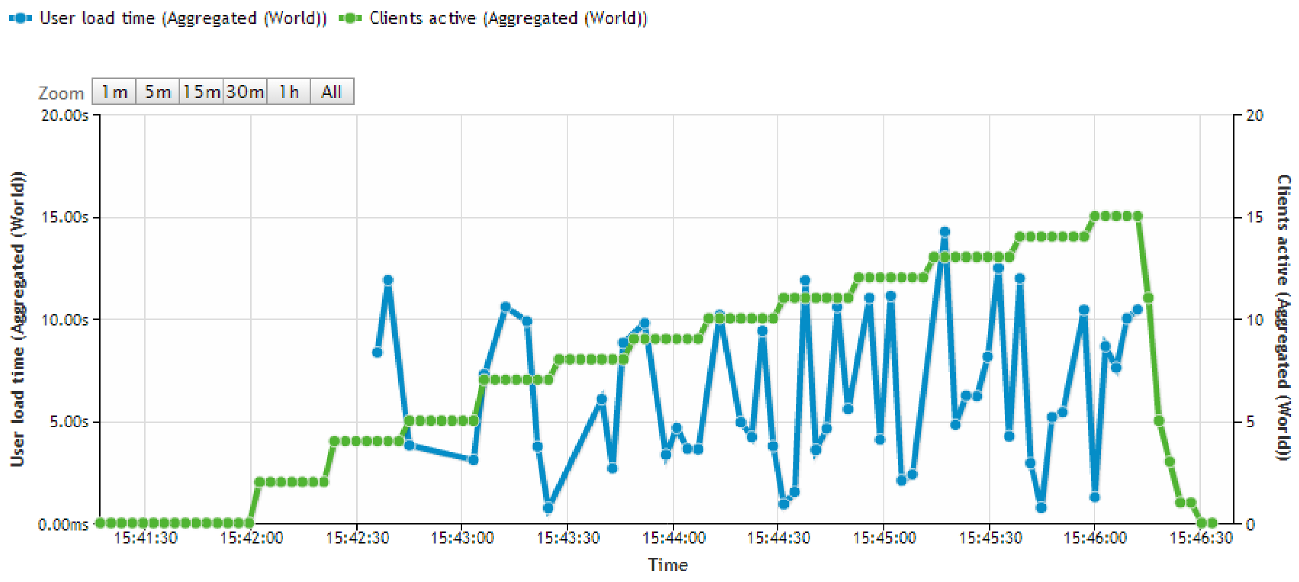


Figure 4. Single server architecture Test case 1

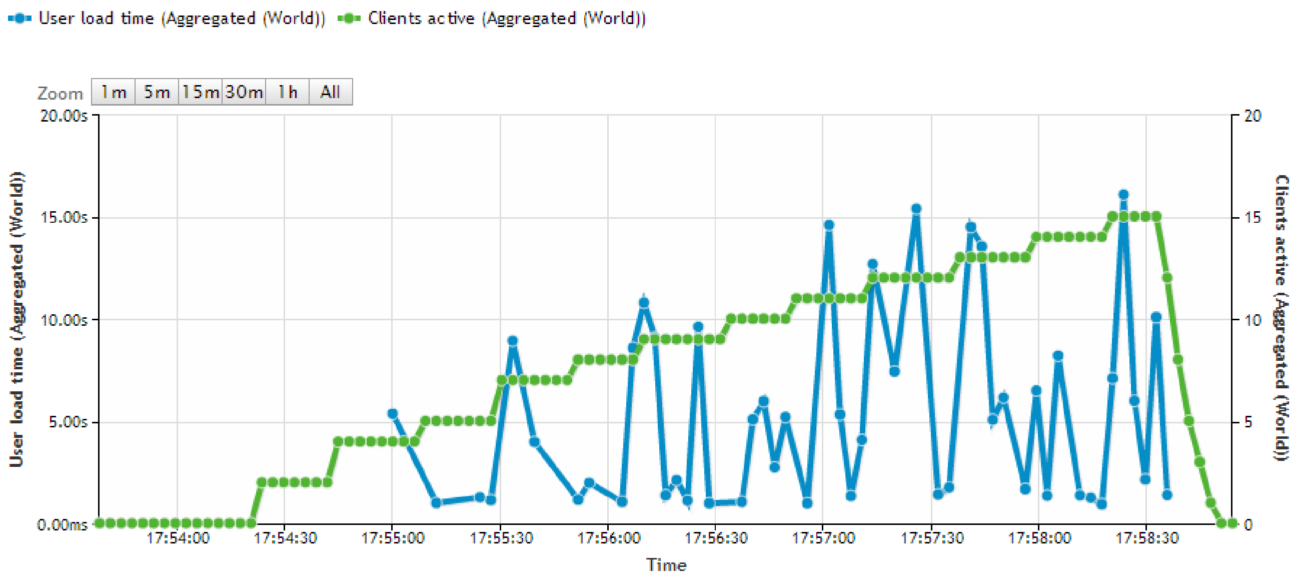


Figure 5 .Load balanced architecture, Test case 1

During both tests, servers handled the load without crashing. As we can see, on both figures user load times vary in the same range. This means that with the average load there is no point in the usage of the new server architecture because it costs more money to maintain without giving any performance advantage compared to the single server architecture.

Test case 2

Looking at the GA statistics, it is shown that on the peak day page visitations (%) were the following:

- 50% of the users were visiting blog
- 30% of the users the front page of the application
- 20% of the users where logging in and doing some actions like looking at their transaction history.

The same user scenarios will be used as in the previous test case.

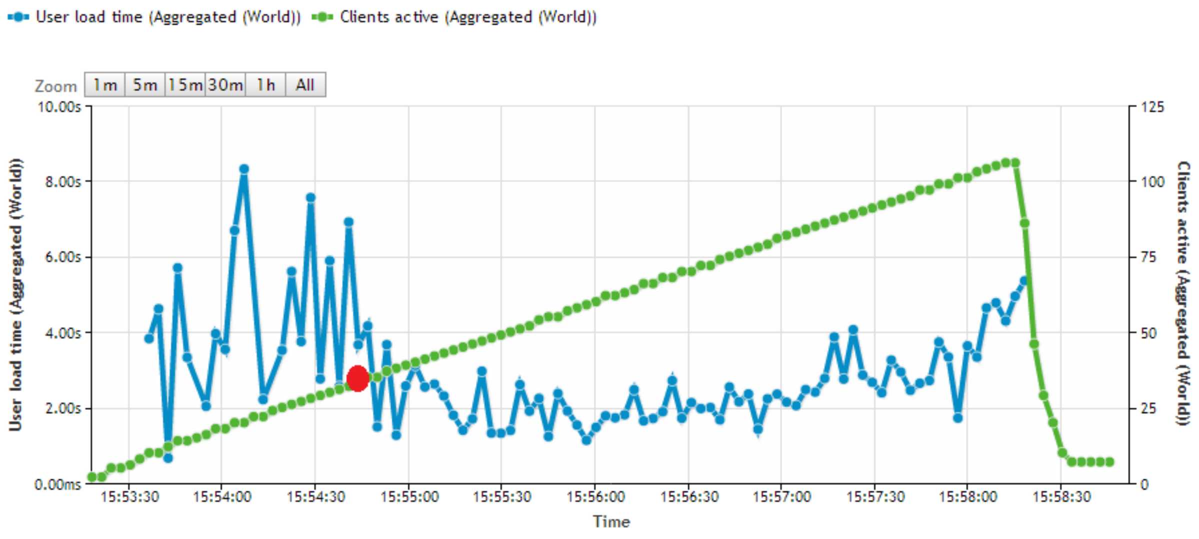


Figure 6 Single server architecture Test case 2

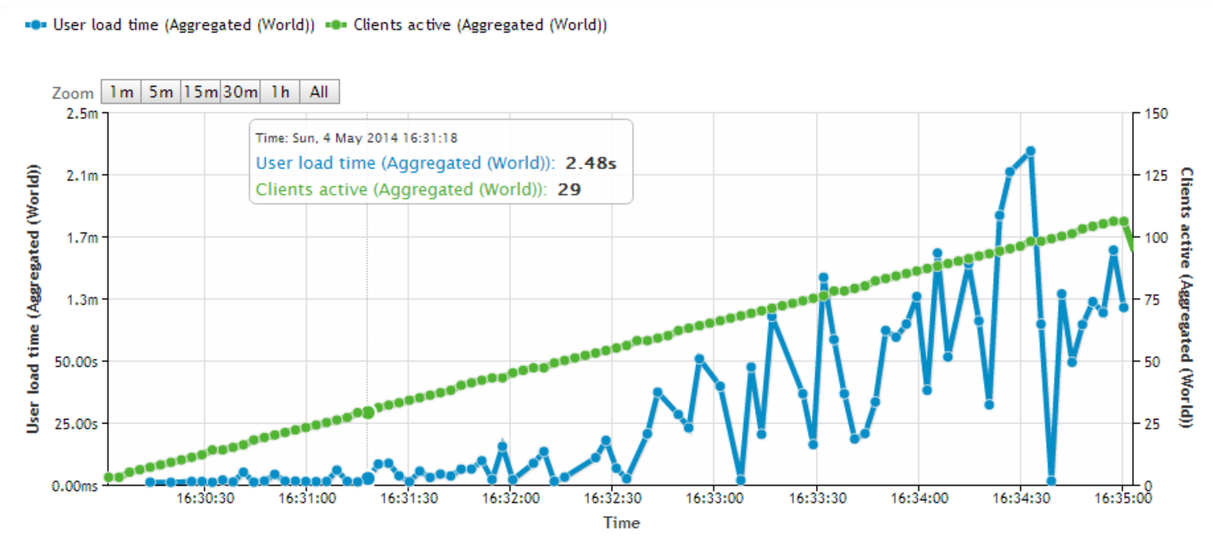


Figure 7 Load balanced architecture Test case 2

During the second test case both of the architectures ran into problems. Single server architecture had significant problems, since it crashed. The crashing point is marked on Figure 6 with a red dot. The crash time was obtained from the Apache server log. With the second architecture, servers did not crash but got extremely slow with the growing amount of concurrent users. As can be seen from Figure 7, by the time the number of concurrent users reaches 75, the load time for user case can take up to 1.3 minutes. Considering that the maximum time for a user scenario in test case 1 was about 15 seconds, it is not acceptable to wait 75 seconds for the users to complete same user scenarios.

Auto scaling

As can be seen from the test results, even three separate servers were not enough to serve 106 concurrent users. Since the average amount of concurrent users is 15 and the maximum amount of users should be around 106, there is no point in wasting money and constantly using the solution that can serve maximum number of users. As briefly mentioned in section 2.2.1, there is a possibility to auto scale servers. In SportID’s case the capacity rises when a new blog post is published. This means that the time of the load increase can be predicted.

Rackspace allows users to create scaling groups. A scaling group consists of the assigned load balancer and servers that can be deployed. Servers will be deployed based on user defined scaling policy. In the policy, user defines by which rules new servers will be added or removed to/from the existing scaling group.

Author considers the case where blog posts are published every Monday at 10:00 AM UTC. After the publishing of a blog post, load increase occurs for up to 24 hours. The highest amount of visits occur on the publishing day, but the load drops to average somewhere in the following day. As the test cases in section 3.1.6.2 concluded, for the average load SportID needs only one server. For the maximum amount of concurrent users even two web application servers were not enough. Considering that the architecture must automatically scale up, initial solution (section 3.1.2) cannot be used. Database needs to be separated from the web application and a load balancer needs to be present in case of extra servers being added. This means that the final solution for 100 000 users on average load will be similar to the one on Figure 3, except it will only have one web application server.

Since the solution presented in Figure 3 was not fast enough in the case of maximum user load, the scaling group will add two extra web application servers to the solution described in the previous paragraph. One of these extra two servers will be removed on the same day. The second server will be removed on the following day. This configuration uses 3 scaling polices: one policy that adds two servers every Monday on 09:55 UTC and 2 scaling policies that each remove one server. Servers will be added 5 minutes before the publishing, because it will take up to 5 minutes for the servers to deploy.

After the two servers were added to the scaling group, author ran the same test as in Test case 2.

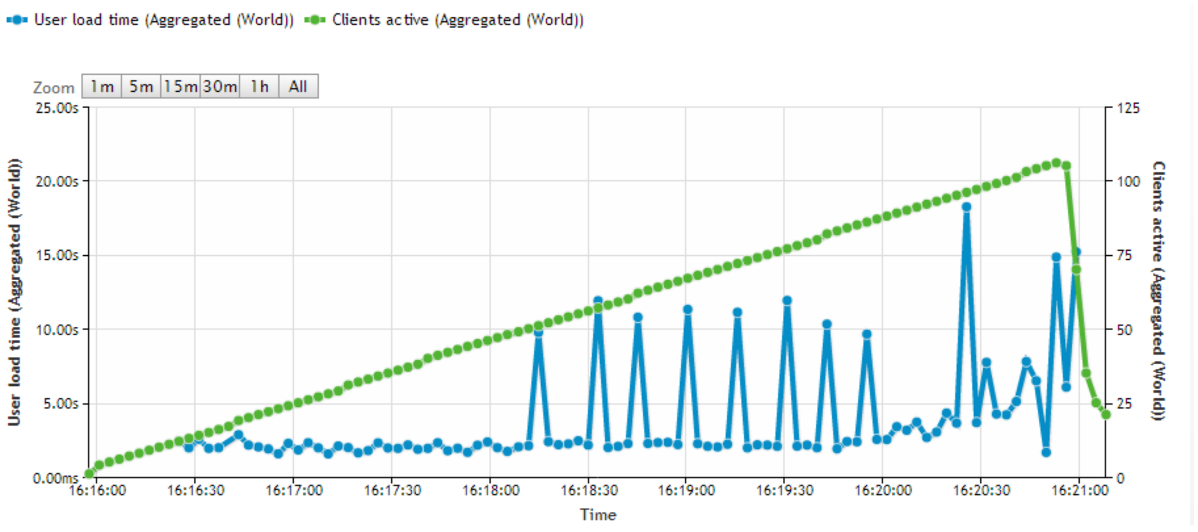


Figure 8 *Auto scale architecture test result*

As can be seen from Figure 8, the results are almost in the same range as in the Test case 1. This timeframe is acceptable for the user scenarios.

Conclusion

When choosing between two different architectures, the one with load balancing definitely has an advantage when it comes to higher amount of concurrent users. Because of the extra resources, it can manage higher amount of users. When we consider 100 000 users as the maximum user base for SportID, the new solution is needed because on a day with higher traffic the regular one server architecture cannot handle the load and will crash. The test results showed that for serving the maximum amount of users the current software solution needs at least 3 web application servers for serving clients.

3.2 Cloud-based data-analysis

Introduction

Today SportID is using GA platform getting statistics about the web page visitation rate and other things. GA is a good place to start since it does not need basically any custom programming to get started. However GA can only analyse certain data. For example it might not know how old a certain web page user is or what age group likes to visit the workout page the most. In the following sections the author is going to test a large-scale data analytics tool for SportID's data analysis.

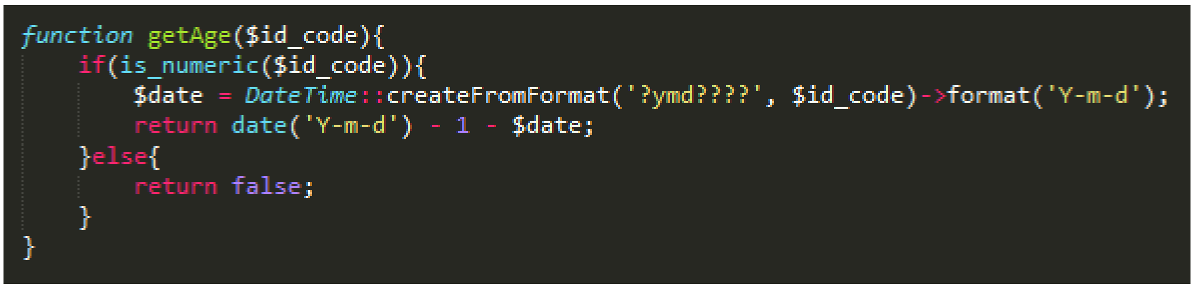
Data-analysis solution

As briefly mentioned in the introduction, there is no way for SportID to know which user groups visit which pages. To know this, we need to find out every page that the user visits and into what age group that user belongs to.

To solve the problem author decided to make a cloud-based data analysis tool that would have minimal cost and would be able to analyse data generated by at least 100 000 registered users. The tool is built on a separate single cloud server that sends the analysed data to SportID's MySQL server. For the data analysis, the author decided to use Apache Hadoop framework (Section 2.4.1) since there is nothing as similar and well documented on the market. On top of the Hadoop framework is set Apache Hive warehouse infrastructure. Hive was chosen because Hive Query Language has the closest syntax to Structured Query Language (SQL) and all of the SportID's developers are familiar with SQL. This means that all of the developers will be able to develop the analysis tool further without having to master a completely new query language.

In order for the analysis tool to be effective, the data that is going to be analysed has to be on the nodes of the cloud analysis cluster. Since this solution is using only one server, the data must be located on that particular server. If the data was held in a separate server, the analysis of the data would take extra time because the data needs to be moved to the Hive warehouse before it is analysed. Also this would be an extra step. It is smarter to write the data to the analysis server right away, rather than save it first on the web application server and then move it again to analysis server.

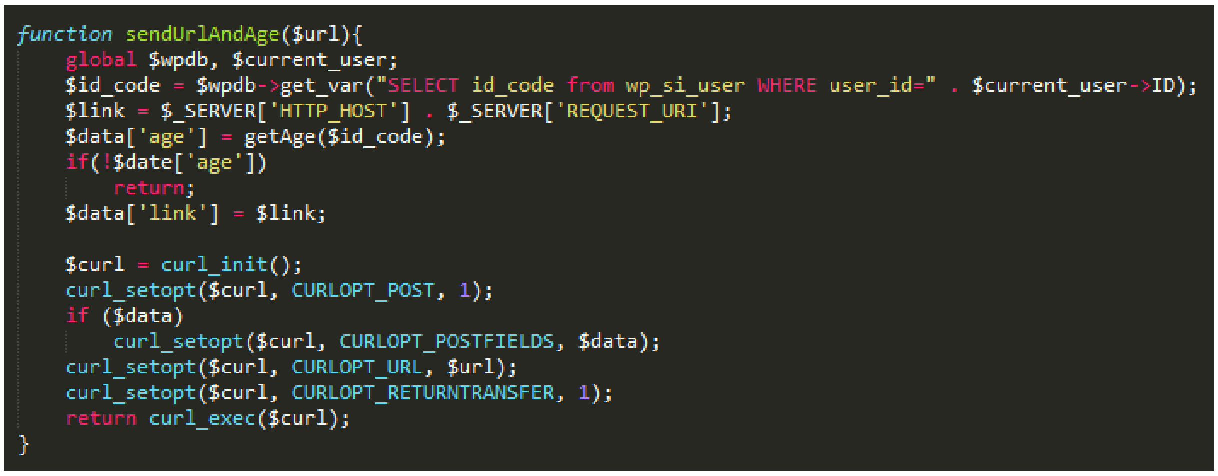
To move the data onto the analysis server, the author had to write a plugin for SportID's application. Since SportID has identification codes stored into the database, it is possible to calculate the age of the user without knowing who the user actually is and therefore not ruining anyone's privacy. To get the user's age, author wrote the following function:



```php
function getAge($id_code){
    if(is_numeric($id_code)){
        $date = DateTime::createFromFormat('?ymd????', $id_code)->format('Y-m-d');
        return date('Y-m-d') - 1 - $date;
    }else{
        return false;
    }
}
```

Figure 9 . Function returning age from identification code

Function shown on Figure 9 takes identification code as input and returns the age if the identification code is numeric.

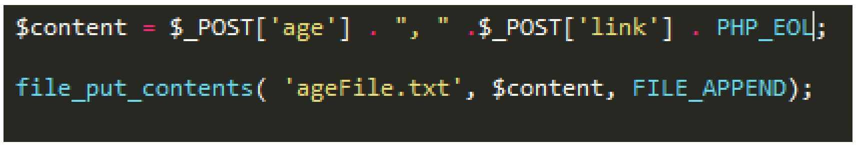


```php
function sendUrlAndAge($url){
    global $wpdb, $current_user;
    $id_code = $wpdb->get_var("SELECT id_code from wp_si_user WHERE user_id=" . $current_user->ID);
    $link = $_SERVER['HTTP_HOST'] . $_SERVER['REQUEST_URI'];
    $data['age'] = getAge($id_code);
    if(!$date['age'])
        return;
    $data['link'] = $link;

    $curl = curl_init();
    curl_setopt($curl, CURLOPT_POST, 1);
    if ($data)
        curl_setopt($curl, CURLOPT_POSTFIELDS, $data);
    curl_setopt($curl, CURLOPT_URL, $url);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
    return curl_exec($curl);
}
```

Figure 10. Function for sending data into remote server

Function shown on Figure 10 finds the current Uniform Resource Locator (URL) and puts it into array with the age calculated from the function shown on Figure 8. This array consisting of two elements is sent to the remote server.



```php
$content = $_POST['age'] . ", " .$_POST['link'] . PHP_EOL;

file_put_contents( 'ageFile.txt', $content, FILE_APPEND);
```

Figure 11. Code for saving visits into textfile.

Code shown on Figure 11 takes the variables from the incoming array and saves these variables into a text file. Each new array is saved on a separate line. Variables are separated with a comma.

Now that the data has reached the analysis server, it can be turned into usable statistics with Hive querying. The following script will create a Hive table that is going to store the data. The data will then be analysed by user groups. These user groups are defined by age ranges. The analysed data will go into a new Hive table that can be later exported back to the database server. Commands are separated with „ ; “ sign.

```
CREATE TABLE new_data(age int, url string) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',' STORED AS TEXTFILE;
CREATE TABLE age_results(age_range string, url string, count int) ROW
FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;
LOAD DATA LOCAL INPATH “/var/www/ageFile.txt” OVERWRITE INTO TABLE
new_data;

INSERT OVERWRITE TABLE age_results SELECT t.range, t.url, count(*)
from (
  select url, case
    when age between 18 and 25 then ' 18-25'
    when age between 26 and 35 then ' 26-35'
    when age between 36 and 45 then ' 36-45'
    when age between 46 and 55 then ' 46-55'
    else '56-66'
  end as range
  from new_data) t
group by t.range, t.url;
```

To move data from Hive to relational database, a tool named Apache Sqoop [17] is used. This tool is designed to efficiently transfer data between different data warehouses and relational database. The following command will transfer the data from the Hive „age_results“ table to SportID’s database table „wp_age_results“:

```
$SQOOP_HOME/bin/sqoop export --connect jdbc:mysql://<IP AD-
DRESS>:3306/sportid --username <database user> --password <password>
--table wp_age_results --export-dir /user/hive/warehouse/age_results
```

Now the analysed data is on SportID’s database and can be shown in a preferred method. The same logic can be used to analyse different types of large data sets.

Cloud-based data analysis over MySQL

As mentioned before, SportID uses MySQL database for storing data. It would be possible to store the unanalysed data in that database and query it for necessary results. However there are few downsides to this.

Considering that the solution should work when there are 100 000 registered users, we can estimate how much page visits there would be every month. According to GA, there were 31 792 page views in the timeframe of April 2, 2014-May 2, 2014. This would create 31 792 lines of data into the database. There are currently 6072 users. With 100 000 users it would make 100 000 * 31792 / 6072 = 523 583 lines of data. With the data generation tool [18] the author inserted 500 000 lines of data with randomly generated ages and pages URL-s into SportID's database table. The table was 31.6MiB large. Because this data is not used by the web application on regular basis it just consumes unnecessary room. 31.6MiB is not a problem for a MySQL database. Since SportID is planning to gather statistics on a longer timescale than one month and on more aspects than the one shown above, the database would get much more data than 31.6MiB.

4 Conclusions and future work

The goal of this thesis was to use load balancing technique on SportID’s web application and to create a scalable cloud-based data analysis tool for future use. For setting up the load balancing architecture, the author used Rackspace cloud environment where SportID’s servers were located. For the data analysis, the author used Hive data warehouse on top of Hadoop framework. Data was analysed using HiveQL.

The author tested different server architectures that would be sufficient for a user base of 100 000 users. The tests concluded that the most suitable solution was to use load balancing in conjunction with auto scaling. With the average load one web application server was enough. With maximum load, three web application servers were needed to handle the load. Thus the conclusion is that SportID’s server architecture should be configured according to the author’s research.

The creation of cloud-based data analysis tool was succesful and it analysed data without using resources of the web application servers. The author came to a conclusion that in SportID’s case it is smarter to hold and analyse data on a separate server because holding the data in the web application database would start taking too much space. Since the analysis tool was written using mainly PHP and HiveQL which is similar to MySQL, the tool can be used and modified by SportID's programmers.

For the future work the author is interested in testing the SportID’s web application SQL queries and code efficiency. It seems that the current application is consuming too many resources and could be written more efficiently in order to use fewer resources. Also the author wants to test the data-analysis tool with bigger amounts of data than 500 000 lines and find out how much data can one cloud server analyse alone.

5 References

[1] Facebook users graph [online] URL: http://www.internetworldstats.com/facebook.htm [Accessed 27 April 2014].

[2] Facebook server park size estimation [online] URL: http://www.datacenterknowledge.com/archives/2012/08/15/estimate-facebook-running-180000-servers/ [Accessed 27 April 2014].

[3] Cloud service provider, Rackspace home page [online] www.rackspace.com [Accessed 27 April 2014].

[4] Article on Target's Big Data Analysis [online] URL: http://www.forbes.com/sites/kashmirhill/2012/02/16/how-target-figured-out-a-teen-girl-was-pregnant-before-her-father-did/ [Accessed 28 April 2014].

[5] SportID's web application page [online] URL: www.sportid.ee [Accessed 28 April 2014].

[6] Bisson, S. 2013, *How I'd build a large-scale web app today versus how I did it 10 years ago* [online] URL: http://www.citeworld.com/article/2114292/development/large-scale-web-app-today-10-years-ago.html [Accessed 28 April 2014].

[7] Amazon Web Services, 2014, *About AWS* [online] URL: http://aws.amazon.com/about-aws/ [Accessed 28 April 2014].

[8] Margaret Rouse, 2013, *Distributed denial of service attack(DdoS)* [online] http://searchsecurity.techtarget.com/definition/distributed-denial-of-service-attack [Accessed 28 April 2014].

[9] Nikolas Roman Herbst, Samuel Kounev, Ralf Reussner, 2013, *Elasticity in Cloud Computing: What It Is, and What It Is Not* [online] URL: https://sdqweb.ipd.kit.edu/publications/pdfs/HeKoRe2013-ICAC-Elasticity.pdf [Accessed 29 April 2014].

[10] Rackspace, 2014, *Pricing* [online] URL: http://www.rackspace.com/cloud/servers/pricing/ [Accessed 29 April 2014].

[11] Amazon Web Services, 2014, *Instance pricing* [online] URL: https://aws.amazon.com/ec2/pricing/ [Accessed 29 April 2014].

[12] IBM, 2014, *What is Big Data* [online] URL: http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html [Accessed 29 April 2014].

[13] Canonical Ltd., 2014, *Ubuntu operating system* [online] URL: http://www.ubuntu.com/ [Accessed 30 April 2014].

[14] The Apache Software Foundation, 2014, *Apache HTTP server project* [online] URL: http://httpd.apache.org/ [Accessed 30 April 2014].

[15] Google, 2014, *Google Analytics* [online] URL: http://www.google.com/analytics/ [Accessed 2 May 2014].

[16] Load Impact, 2014, *Load balance testing tool* [online] URL: http://loadimpact.com/ [Accessed 4 May 2014].

[17] The Apache software Foundation, 2014, *Apache Sqoop* [online] URL: http://sqoop.apache.org/ [Accessed 4 May 2014].

[18] generatedata.com, 2014, *Automatic random data generation tool* [online] URL: http://www.generatedata.com/ [Accessed 4 May 2014].

[19] Amazon, (2011), Auto Scaling Architectural Diagram [ONLINE]. Available at: http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/image/simple-as-process.pngWhatIsAutoScaling.html [Accessed 27 April 2014].

[20] Satish, Narayana Srirama (2013), Large-scale Data Processing on the Cloud, Lecture 2 Available at: https://courses.cs.ut.ee/MTAT.08.036/2013_fall/uploads/Main/slides2.pdf [Accessed 28 April 2014].

License