Optimizing performance on current hardware

An investigation how to improve the performance on a slow database without the need of buying extra servers.

S. W. Vendrik

Easysize IVS
København, July 1st 2016

Project leaders:
David Babayan (Lead developer & CTO Easysize IVS)
Gulnaz Khusainova (Founder & CEO Easysize IVS)
Berend Wilkens (supervising lecturer & 2nd examiner, Hogeschool Utrecht)
Roel Veldhuizen (1st examiner, Hogeschool Utrecht)

Version 1.0.0

Revision management

Version	**Date**	**Changes**
0.0.1	May 10 2016	Document has been created and placed into Subversion
0.0.2	May 11 2016	Added chapter 1 and 2 from Plan of Approach into the document
0.0.3	May 12 2016	Added organizational chart
0.1.0	May 13 2016	Added chapter 3: main questions, sub questions, assignment
0.1.1	May 16 2016	Added chapter 4: research methods. Updating table of contents.
0.2.0	May 17 2016	Added chapter 5 to answer sub question 1. Added tier model
0.3.0	May 18 2016	Minor changes to chapter 5, scrapped question 2, start with chapter 6.
0.3.1	May 18 2016	Forgot to update version number, added conclusion of sub question 1
0.3.2	May 19 2016	Finished answer question 2.1, cleaned up a little more and added graph to §6.2 for completeness
0.3.3	May 20 2016	Polishing up last nasties and killing grammar mistakes.
0.3.4	May 20 2016	Chapter 5 paragraph numbering was messed up; fixed that last-minute.
0.3.5	May 21 2016	Forgot to update version number! Sloppy! (no ninja-fixing this time, new revision)
0.3.6	May 23 2016	Forgot to mention Opbeat, added it.
0.4.0	May 31 2016	Processed feedback from 1st concept. Version 0.3.4 was delivered as concept.
0.4.1	June 02 2016	Added deployment model, updated table of contents.
0.4.2	June 02 2016	Updated organizational chart, after a massive employee change on 31st of May and 1st of June.
0.4.3	June 02 2016	Messed up figure numbering, fixed.
0.4.4	June 03 2016	Updated explanation on dashboard, last outstanding remark.
0.4.5	June 06 2016	Added 3 months money back policy
0.4.6	June 06 2016	Fixed misunderstanding: Company does not have its own servers; instead they are rented from Microsoft Azure.
0.4.7	June 07 2016	Forgot to update version number (again), and finally removed a fatal grammar-based typo in chapter 6 title.
0.5.0	June 07 2016	Start with §6.2: current and expected load on the server
0.5.1	June 09 2016	New metrics came in, updated.
0.6.0	June 09 2016	Added chapter 7. Disappointingly short...
0.6.1	June 09 2016	Finished calculations §6.2
0.6.2	June 10 2016	Further worked out §6.2, fixed calculations
0.6.3	June 10 2016	Fixed date on report, fixed calculations and removed the old ones.

0.6.4	June 10 2016	Fixed a number of mistypings and grammar mistakes
0.6.5	June 20 2016	Processing remarks from 2nd version
0.7.1	June 21 2016	What are the most cases of slowing down? Answered (0.6.5 was meant to be 0.7.0 actually)
0.7.2	June 22 2016	Updated Chapter 8 with actual statistics, start to explain solution
0.7.3	June 22 2016	Added CTO role to David Babayan (title page)
0.7.4	June 22 2016	Chapter 8 finished, time to answer main question
0.7.5	June 22 2016	Chapter 6, statistics, made it more “statistic” (less accurate, makes it easier to estimate future developments)
0.8.0	June 23 2016	Added chapter 9 to answer the main question.
0.8.1	June 23 2016	Finished chapter 9
0.9.0	June 27 2016	Added chapter 10, describing of the results.
0.9.1	June 27 2016	Now also with justification of the set requirements
0.9.2	June 28 2016	Added the ultimate objective, it was not mentioned clear enough.
0.9.3	June 29 2016	Almost everything should be there. Time for a spell check, the first two chapters and heading to final version.
0.9.9	June 30 2016	Checking for grammar mistakes
1.0.0	June 30 2016	Grammar mistakes fixed.

Management Summary

Cause

Easysize is a small technology company offering a size calculation tool for online shops. The company is successful, but their servers are reaching the top of their capacity. The company wants to overcome this problem before they are getting too big. While they were first looking into load balancing, it was too expensive to do it. Instead, the focus was moved how to get more performance from the current hardware.

Recommendations

The recommendations made is mainly moving away from MySQL and choosing MariaDB or PostgreSQL instead, where PostgreSQL is preferred.

The company is currently trying to solve the problem currently by switching from MySQL's InnoDB storage engine to the older MyISAM. This seriously threatens data integrity, since MyISAM is sensitive for bad things like a power outage, causing data loss and possible table corruption.

The last serious recommendation is to move the database to a separate machine with no other running things than the database, so it can take up as much RAM and CPU cycles it needs. In the current situation, they have to share it with the nginx web server and the uWSGI application server, as well as some other applications.

Motivation

Initially MySQL was researched and there were no real further optimisations that could be made. The configuration was correct, and indexes were made on places where it makes sense.

One of the possibilities tested was to move data once a day to a temporary table in the memory, but that approach did not improve the performance, unfortunately. Therefore, a look at other systems was worthwhile

MariaDB and PostgreSQL came up as potential replacements. MariaDB is a fork of MySQL, but has many improvements on speed and the optimizer. It profits from its MySQL heritage, and is therefore easy to replace. However, MySQL and MariaDB are moving away from each other, so switching systems may become harder in the future.
PostgreSQL is a different system, with its own history and SQL dialect. The application needs to be updated and the tables moved to the new database. For the database migration tools exists. Clear migration instructions are provided in the report to make the update to go as smoothly as possible. However, it is worth it considering the advantages.

To keep the performance up, the databases may need to be monitored for a while and tweaked a bit further. The systems operate differently anyway (especially PostgreSQL) and the default configuration is rarely the best.

Consequences

However, there are a few drawbacks that need to be considered. A database change is of course a major operation that has to be planned thoroughly. Backups must be taken, the new software installed, moving the data to the new database, cleaning up and tweaking the configuration and application. This makes a downtime unavoidable. The best moment to do this is at night to make inconvenience as little as possible.

The employees of the company have to get familiar with the new SQL syntax, to avoid writing SQL code in a way the database does not understand. The ORM used in the web application partially solve this problem, since it can generate code for many databases, including MySQL/MariaDB and PostgreSQL.

A difference approach of database management is needed, especially PostgreSQL, because its logs sometimes offer performance improvements. They have to be read frequently to find anything interesting and boost the performance.

Preface

With this investigation, I will finish my internship. During the search of a final internship, I wanted to do something more than the standard. I decided to move out of the Netherlands to another country to finish my course. After a number of companies I ended up at Easysize, which is situated in Copenhagen, Denmark.

At first I it was planned to help the organisation with a number of the programming tasks. They had an extensive roadmap showing them where they want to go. I would be a supporting person in it. However, during the first days, it became clear the company was facing a more serious problem.

The servers were reaching the maximum of their capacity, and needs to be sorted out. This fitted good in what my institute, Hogeschool Utrecht, was expecting. A research was set up and I began digging into the servers and the configuration.

The company provided me with my own “test server” to test any theories and solutions I had found. This was a useful extension that allowed me to reinforce any decisions I took.

Finally, I would like to thank the company for the trust they had in me and offering an excellent opportunity to finish my course. I also want to thank David Babayan for being my company supervisor and helping me to bring my investigation to a good end.

København, 2016 July 1st
Sebastiaan Vendrik

Table of Contents

1. Introduction

It is a yearly recurring phenomenon: The Dutch tax declaration. Every year, their website tends to break down under the heavy load. Too many people try do send their declaration at the same time. Also, processing these declarations take a long time. On the Dutch news site nu.nl (http://www.nu.nl) reporter Poort (2014) wrote an article, stating the Dutch tax authorities would give everybody 5 days extra to do turn in their tax declaration. The tax declaration site was often hard to reach (sometimes even completely unreachable), forcing them to give citizens this extra time. At the end of 2014, the Dutch government decide to give everyone a month extra (to turn in their declaration of next year, starting on the 1st of March 2015), because they did not manage to fix their computer systems in time.

Unfortunately, this situation is not unique. Many organisations suffer from outages of their systems for all kinds of reasons. Sometimes, it is just a system malfunctioning, while in other cases, systems are simply getting overloaded by excessive interest of the public, as described in the first paragraph. To make matters worse, websites are sometimes intentionally brought down using a Distributed Denial-of-Service (DDoS) attack: Hackers try to generate so much traffic to clog up the connection lines to the server, so the servers cannot respond to any legitimate requests any more. It is technically a traffic jam on the digital highway.

Easysize IVS is a small company who offers its service to online clothing retailers. Easysize offers an API to extend web sites from retailers with a clothing size calculation tool. The company is growing, and to be able to deal with the increased load on its API servers, it needs to know what their servers and API can take, and more important, figure out how they can balance load between multiple servers so they won't go down when their use base (or one of their customer's) grows.

This thesis is the result of the investigation that has taken place in the past few months. It describes the way the investigation was done, the (main) question that was to be answered, the break-down of the main questions into sub questions to investigate to finally answer all the questions.
Finally, the results of the investigation will be revealed, as well as an advice report and a working proof of concept.

2. The company

2.1 Easysize IVS

Easysize IVS is a technology company based in Copenhagen, Denmark. The company has developed a technology that predicts the correct clothing size for online shoppers. The aim of the project is to predict the right size of clothing for online shop customers, in order to decrease the number of returned clothing. It also decreases the costs and time the online shop has to incur because of the returns.
The customers, on their turn, will get more confident while placing orders, because they don't have to worry if it will fit in the end.

According to their web site (http://www.easysize.me/) returns will drop with 35 – 40%, while some customers report an increase in sales with 25 – 42% at the same time.

A recent change in the business model allows the company to make money per predicted size. Shops can order a subscription, with a set maximum of predicted size each month. As retrieved from their web site (http://www.easysize.me/price) the cheapest option is a maximum of 5.000 predicted sizes for €49,--. Both the predicted sizes and costs are per month, so the simplest plan will grand you a total of 60.000 predicted sizes a year for €588,--.
Previously, we also offered a one-month free trial to web shops, this has been replaced by a three-month money back guarantee. If a shop is not satisfied with our service, they will get a refund for the three months they tried our service.

If a shop would like to have more than 200.000 predicted sizes a month, the shop has to contact us though the contact form so the company can provide them with a custom packet that fits their needs, with a custom price.

The previous business plan still applies to existing customers who had a subscription before this change. On the old web site (https://web.archive.org/web/20160320151145/http://www.easysize.me/price/) there are 5 plans offered, where Easysize would be paid for each order made with an Easysize predicted size rather than only predicted sizes.
The simplest plan was limited to 50 orders a month and was available free of charge. The most expensive plan however, did not have a set price but would be decided on the needs and interest of the shop. An unlimited amount of orders could be made with the most expensive plan.
A free trial period of one month was available.

The service also comes with a dashboard for customers, providing insights about their shop and customers. It provides features like user counting, number of predicted sizes, the number of purchased items (after size predictions), the customer location (country) and numerous other statics that may be interesting for the shops.

Easysize IVS is a small company currently consisting of 8 people. Because of the small and young nature of the company, the workforce tends to be volatile and can change rapidly. The described

Organizational chart is based on the workforce as it was on the 1st of June, 2016.

Easysize IVS is a small company currently consisting of 9 people. The company has currently 2 divisions, product developed and sales. It has no separate staff division. The intern will work at the product development division.

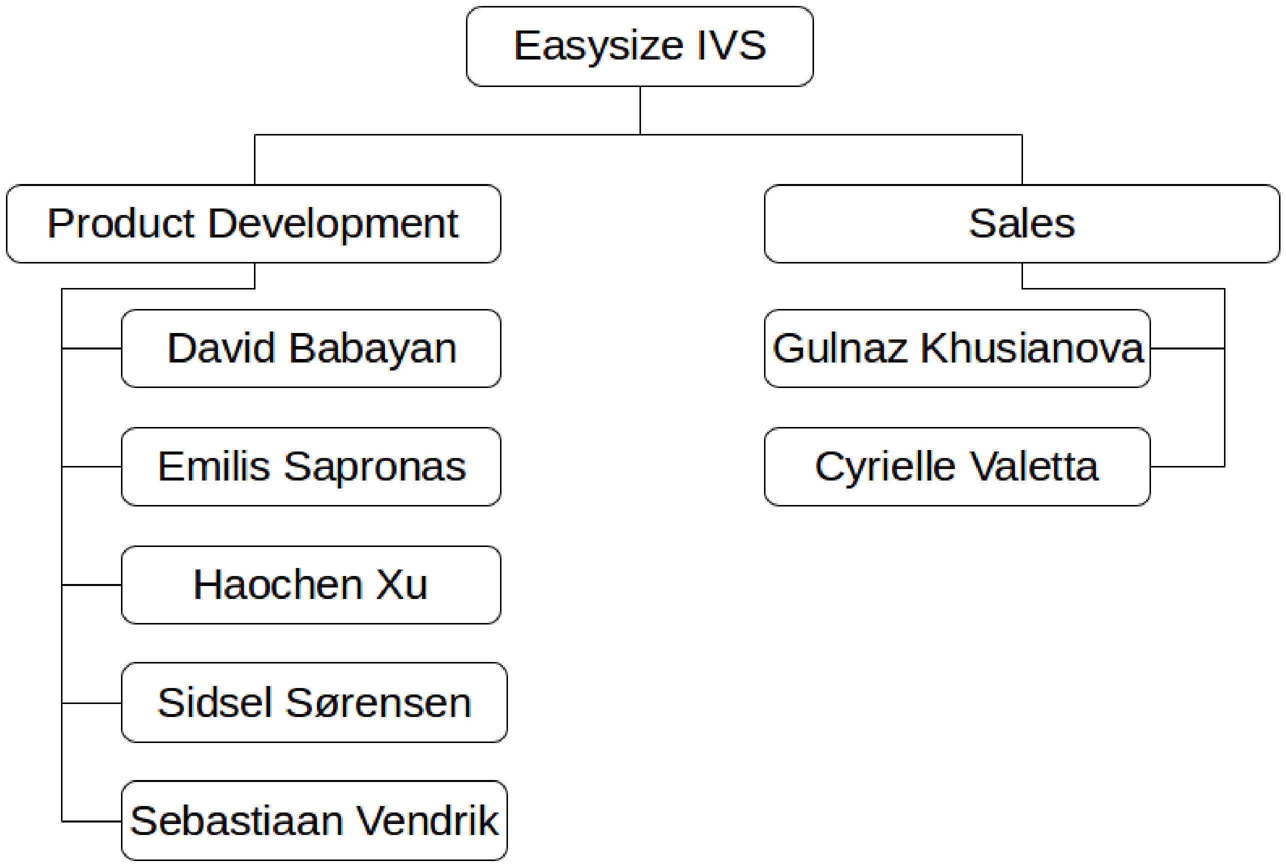


Figure 1: Easysize's organizational chart, showing the divisions and employees

David Babayan is the lead developer of Easysize's software solutions and will be my company supervisor. The intern will do all that is possible on his own, but can ask questions to David.
David will review the research and found solutions, give advice on the company's position and the feasible options, now and in the future.
A review moment takes place weekly at Friday, and implicit every morning during the *Daily stand-up* (comparable to the daily SCRUM talk).

2.2 The place of the trainee

The company has a small development team consisting of 4 people. There is a sales team too, it currently also consists of 4 people. The intern joined the development team to improve the company's product. He figured out the current infrastructure set-up and investigates where (potential) bottlenecks were going to be, addressing them before they could get too serious.

The final goal was to make a test set-up to show where the infrastructure and construction was going to fall short, and what options were taken to make sure the service could keep up with demand.

2.3 Expansion plans

Easysize IVS is still a small company, but has ambitious expansion plans.
The company has currently a contract with Microsoft Azure, which provides virtual servers. Easysize uses these servers to provide their services. The services were handled by two co-operating servers without any load-balancing whatsoever. The company needs to keep its customers happy, so the service has to be fast, reliable and keep up with demand. If the service slows down, it tends to slow down an entire customer's web site.

The service consists of 3 tiers: The client tier is technically the customer's web site, accessed by a browser. The Web/App tier runs Nginx as HTTP server, uWSGI as Python application server and the web application itself is written in Python on top of the Django framework. The last tier is the Database tier (Sometimes called EIS tier). The database is managed by MySQL 5.6. The servers operating system is Ubuntu 15.10 Wily, the MySQL software originates from the Ubuntu software repositories.

The company gets a few new customers each month, but wants to expand faster.

2.4 Goals

The company now has a number of options they can implement to cope with demand for the coming years, allowing them to grow. The options are different, so the company can adapt to a number of future scenarios. They do not have to spend much money on a solution that may be too big for it right now, but can be considered later.
The possible options have been demonstrated with a proof-of-concept, and are comparable with the previous situation. The service will keep up with demand.

3. The problem

The starting company wants to grow. More customers mean more money, but also a higher load on the production servers. To keep the customers happy the service needs to be fast. Furthermore, at the beginning of May it was announced Easysize was invited to attend an exclusive investor event in Norway, as one of the most promising start-ups of Copenhagen.

Especially the last mentioned event can generate a big prestige boost, attracting more customers to try (and keep using) our services. Therefore the service needs to remain fast. What means “fast”? Is a response time of 2 seconds fast? What about 200 milliseconds? Are there any other “performance” things other than time? These are all questions that have to be answered.

3.1 Main question

To define a clear assignment, and to provide a clear direction, there is a need for a main question that is overarching the whole case. After great deliberation, the following main question was conceived:

How can the service and its infrastructure be improved to maintain the performance requirements?

To answer this question properly, it is broken down into a number of sub questions. These will be answered first and their answers will automatically the main question.

3.2 Sub questions

1. What is the current service set-up?
   1.1 What are the service & API performance requirements?
   1.2 What options does the current set-up offer?
2. Which threads are currently there to the service performance?
   2.1 What are the company expansion plans?
   2.2 What is the current and expected load on the service?
   2.3 What are the most common causes of slowing down?
4. What measures have already been taken to increase the maximum load?
5. Where are the current and potential bottlenecks and how can they be fixed?

Once these questions are answered, we can start looking for possible solutions to finally answer the main question.

After much consideration, the original question 2 (*What are the service & API performance requirements?*) has been dropped. The main question is now integrated as sub question 1.1, while the sub question (*Which products does Easysize use for its API?*) has been removed at all. Question 1 already mentions the used software products and their versions, rendering sub question 2.1 obsolete. With just a small question remaining that relates directly to question 1, so the decision was made to put them together.

During the writing, another sub question for question 1 emerged: What options does the current set-up offer? Instead of fixing our attention to one point or solution with more servers (for example), it may be easier and cheaper to see what can be done with the current hardware we already own. Maybe it is just in need of some optimisation.

Originally there was a question 5.1: *How is load testing done at the moment?*
This question has been removed because it does not help to answer the question where our potential bottlenecks are; it rather answers the question how they are going to be found and identified.

4. Research methods

During its course the intern has learned three different ways to do research. These are known as a quantitative market research, desk research and literature review. Depending on the question and the needed answers, the intern will decide which of them is most suitable. Furthermore, these three methods won't be sufficient and will require digging into the infrastructure and software to get familiarized with the systems.
Since there will be no contact with customers, the quantitative market research will not be carried out.

4.1 Desk research

Desk research uses the results of the market research from other vendors. It will be a waste of time and money to re-research already studied problems.
Using this way of research it is possible to quickly find results and refine them to show only the data you want, and make a prediction of how the market is likely to change.

4.2 Literature research

This way heavily relies on books and internet for information. This technique used to find existing information about a particular subject. Many used standards or standard methods are well documented, and using this research we can decide how it is usually done and motivate why.
This one is particular interesting because it also delivers information about the used systems and software. To get familiar with the system the documentation is needed to get a better understanding, and to discover what options are available.

4.3 Research methods per sub question

Question 1: Since the system is build after a certain design the documentation about the project will have to be read and understood. This is very likely to be a literature investigation. If there are any gaps left, an interview with David will be organized to fill in the blanks.

> *Subquestion 1.1:* This will be a combined technique using both literature research and meetings with the developers. The current goad and plans have to be figured out. This question is helps to complete the report to describe the current plan and the goal that the company has set.
>
> *Subquestion 1.2:* The information gathered in the previous question will be revised and the options investigated. The possibilities will be tested on a specially set-up testing servers to see if the hypothesises were right and to discover when it is usable in the company.

Question 2: This question can only be answered by cutting it into a few sub questions and answer them instead of trying to figure this one out as a whole. Therefore, this question is too big to scale it down to just one method.

Subquestion 2.1 This will be most talking with the sales division. The sales division has a view on what they are trying to achieve, their goals and the number of new customers they got in the past few months.

Subquestion 2.2 This question can be answered with mainly calculations based on the information acquired in the previous (sub)questions, together with some of the current load testing reports that have been already made by the developers.

Subquestion 2.3: This will be a combined literature and desk research. The theoretical limits have to be discovered, and the most common cause for the slow down problems, for example high database latency. By combining the gained knowledge the limitations that will be hit first can be determined, and be dealt with.

Question 3: This will involve another talk with the developer and the system operator. Also the documentation of the used tools will be also investigated if there are any. If there are no tools (for example; load balancing) used, research will be done to find out what kind of solutions are available.

Question 4: This will be a combined research of the project's literature and the actual set-up, because the weakest link on the internet determines the speed. The potential bottlenecks will be investigated and see if there come any unforeseen problems popping up there.

5. The current infrastructure

The preliminary investigation has been completed. In the next weeks, the current set-up settings have been reviewed and an extensive research has been carried out. There is enough data right now to draw conclusions. The research results in the sub-questions being answered, which will ultimately answer the main question.

5.1 What is the current service set-up?

The service consists of three server-side parts, and the client side. The client side is a browser, either on a mobile device or a desktop computer. When it connects to an Easysize–enabled web shop, a simple JavaScript embedded on the web page points the browser to make a connection to Easysize's web service.
The UI elements presented on the page are programmed using HTML, JavaScript and CSS. The server's back-end is written in Python 3, using the Django 1.8 framework. Data is stored in a database. Communication between the website and the API server is done using HTTP, with the REST architecture.

The HTTP server is provided by Nginx. The official documentation (nginx Inc, 2014) states it is written as a multiple purpose proxy server. Nginx is capable of “reverse proxy server, a mail proxy server, and a generic TCP/UDP proxy server” (retrieved from http://nginx.org/en/, section 1, paragraph 1). In the company's use case Nginx is utilized as a reverse proxy to uWSGI.

uWSGI is an application server written in C. The documentation provided by Read the Docs and uWSGI (2014) says the application server currently supports Perl, Ruby and Python as programming languages. The server itself can also support “plug-ins” written in C, C++ or Objective-C. uWSGI is currently used to run our service, which is written in Python, and the Django framework where our application is build on.

Django profiles itself as a Model-View-Controller framework for Python projects (Django Software foundation, 2015). Django itself is also written in Python. Django can be divided into two separate frameworks the “standard” Django and Django REST framework. The latter one makes web applications available using the REST standard. The difference between the two is the “standard” Django only produces HTML, while Django REST produces JSON. The application uses various API's provided by Django, including the ORM that comes with it.
According to Emilis Sapronas (personal communication, May 3rd, 2016) Django also features a build-in ORM to generate most of the queries. However, David states some complex queries (mostly the ones with multiple joins) have to be written by hand in the code.

The database currently in use is MySQL 5.6, with InnoDB as default table engine.
The performance server-side is measured using Opbeat. The official documentation (https://opbeat.com/docs/articles/dashboard/) explains Opbeat is a service that can calculate how much time an request needs to complete, and divides time into the portion used by the application itself, the database, the cache, templates and time used by external (third party) services.

With all of this information, a tier model can be set up. A tier model defines an architectural deployment style. The tier model describes the separation of functionality into segments, and on top of that the tiers can be placed onto the same physically computer, but can easily be put onto separated computers when needed.

Based on the current set-up, we can either define a 3 tier model and a 4 tier model. In the 3 tier model, the web server and application server runs on the same tier, where a 4 tier model separates these two things on separate tiers.

After an interview with David Babayan, the 3 tier model described was confirmed: There is no separate client (apart from the browser) and therefore there is no need to split the app server and web server into separate tiers.
Figure 2 depicts the standard 3 tier model the company currently uses.

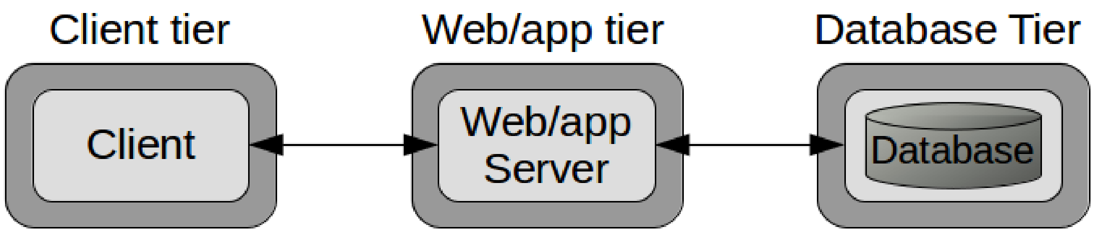


Figure 2: The current tier model

The client can be either a computer or mobile device, while the web/app tier and database tier are put on the same server. A corporate user buying the most expensive packet (see §2.1) will get their own server, with their own web/app tier and database tier.

The system works in the following manner: The client first visits an online web shop running Easysize. The browser gets redirected on the background to the Easysize Web/App server and retrieves a number of scripts and code over an HTTP connection to provide the service. When a size calculation is requested, the client contacts the API server over a secure HTTPS connection, which does the calculations and has the database. Figure 2 contains a deployment model for further clarification. Note there are 2 servers involved in the Easysize service life cycle.

Finally, there is also a 3rd tracking server. This server does not take part in the size calculation process, but tracks the user to provide the data for the customer dashboard (see §5.2). Things like users, predicted sizes and purchased sizes are tracked by this server.

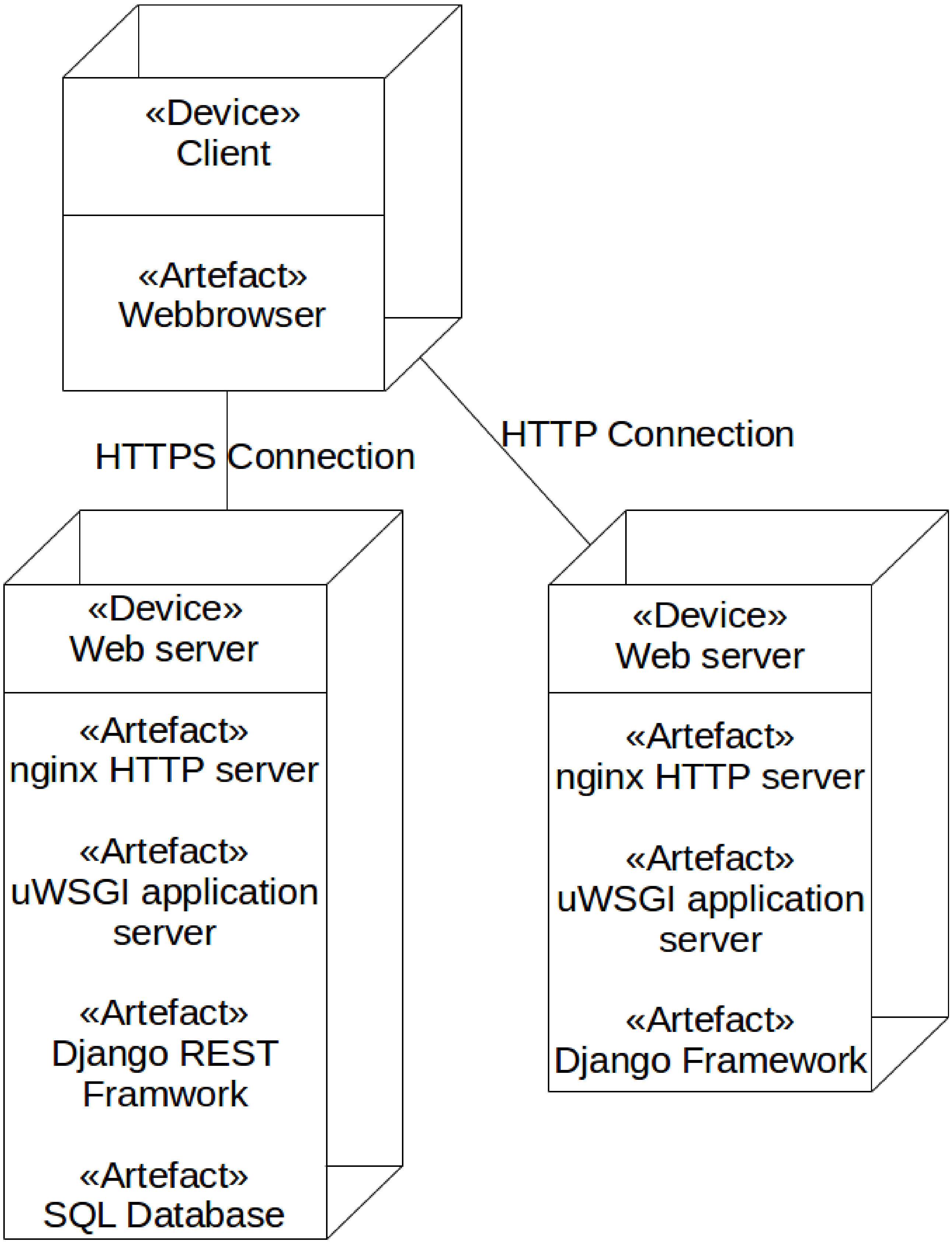


Figure 3: The Company's deployment model

5.2 What are the performance requirements?

The ultimate goal is to have the service to take at most 300 milliseconds. There has no set time slices for any of the tiers or used programs. The network latency does not take part in this network connection, since the connection changes from day-to-day and Internet Service Provider.
At the moment of writing (May 20 2016), a number of improvements have been implemented, causing the API to respond in time (excluding the latency). Only the customer dashboard (see §2.1) and internal dashboard (comparable to the customer dashboard, but with more statistics interesting for the company itself) is still too slow.
The performance is measured with Opbeat. It will tell us how much time the application spends without taking network latency in account. Browser measuring for example does count network latency.

Although almost all requirements are met at the moment, the improvements are based on a number of conclusions from my research. More long-term solutions should also be investigated and recommended for future use.

This is also the ultimate objective of the whole research. While it still performs in a timely manner, performance issues starts to arise and the company wants to address them to prevent them from becoming too big. Also, one of the chosen solutions is to move bad performing tables to MyISAM, which threatens data integrity. Moving tables to MyISAM should be considered an emergency solution for a couple of days, but not as a permanent solution.

5.3 What possible options does this set-up offer?

The fact that Nginx is used as a HTTP server, offers the possibility of easy-to-implement load balancing natively. The Nginx documentation (Nginx Inc., n.d.) explains how Nginx is able to load balance between the requests and how to implement this. The fact the database is on a different tier makes it possible to move it to another, separate computer. Database operations can take relatively a lot of time to execute, especially disk access is needed (for example, after a transaction has been concluded with a `COMMIT`) or needs a lot of CPU cycles (in case the database need a lot of calculations to execute on the retrieved data).

The Django ORM web site (Django Software Foundation, 2015) shows it supports 4 databases; these are PostgreSQL, MySQL, SQLite and Oracle. All databases needu their own connectors. There are also a number of unsupported databases (i.e. Microsoft SQL Server) that can be used with plug-ins from 3rd parties.

The use of an ORM has the added benefit a database switch is easy. However, the raw queries in the code prevent us from doing so. David Babayan explained the Django ORM has some limitations, it cannot natively JOIN tables. The raw queries are a way to work around that limitation.

5.4 Conclusions sub question 1

After the research, it is revealed the service is written in HTML, CSS and JavaScript for the website front-end. The back-end is a so-called RESTful web application, where the Nginx server handles

the HTTP requests. They are forwarded to uWSGI, an application server made for python web applications (while uWSGI supports other programming languages as well).
The actual API code is written in Python version 3, with help of the Django framework. Django also offers an ORM, which is used to generate some of the database queries. The database used to store everything in is managed by MySQL.

The fact the company decided to program in Python 3 from the start is an important decision, since Python 3 is incompatible with the current standard Python 2. While many Linux-systems come with both Python 2 and Python 3 installed today (own experience), the website *The Hitchhiker's Guide to Python* (http://docs.python-guide.org/en/latest/starting/install/osx/) states that Mac OS X still ships with Python 2.7 by default. After asking David Babayan about this, he confirms Mac OS X still ships Python 2 standard installed, but not with Python 3, and this even counts for Mac OS X El Capitan.
However, this decision makes the application future-proof, since it makes sense Python 3 will probably longer supported than Python 2.
The fact they use an ORM for database management gives them more freedom to choose the database they want. However, since there also hand-written queries in the code, this is quite a hard thing to do right now.
The Nginx HTTP server offers some potential as well. It is currently used as an HTTP server and reverse proxy to direct the requests to our API. It is also capable of load-balancing requests between multiple servers if that turns out to be an option.

The tier model is currently based on a 3 tier model: Client ↔ Web app ↔ database. Because there is no stand-alone client, a 4th tier does not make sense. With the current model, the database could be moved to a dedicated server, since a database ideally gets much ram and a lot of CPU cycles.

6. Threats to performance

6.1 What are the company expansion plans?

For this data I've had an interview with Anna Chan and Gulnaz Khusainova, both leading the sales division. To acquire the necessary data, I've made a total of 4 questions to get started.

1. How many customers does Easysize serve at the moment?
2. Where can I find the results of previous months
3. What goals have you set for the upcoming period?
4. What are the expansion global expansion plans?

The interview with Anna took place at May 18th 2016.

During the interview it turned out the company has multiple definitions of “customers”. It is used for both companies that have Easysize integrated, as well as customers who calculate their size, and finally unique visitors on a shopping page. In this document, customers will only refer to the web shops that have Easysize integrated.
Anna stated Gulnaz can give the exact numbers. However, at the moment 7 shops are currently using the Easysize service. 5 shops are paying customers; 1 shop is currently negotiating for contract renewal, while the other one is still in its trial period.

She didn't know any data of the previous months or where to find them. Carl Johan Rising can help, since he is the data scientist.
After asking the same question to Carl Johan, he explained the data can be found in the weekly newsletters David Babayan sends out every week. The weekly statistics are in there, together with the monthly statistics.

She explained the company works in phrases of 6 months each, stretching from January until June, and from July to December.
For the first period of 2016, the current goal is to have 15 shops in total. Having the current number of 7, the company is behind schedule. However, many shops will start offering Easysize's sizing services from June, so there will be more shops at the end of the period.

For the second period, the company's goal is to attract at least 19 new customers.

Finally, she revealed Easysize is currently expanding to more countries. While they started at the Danish marked, their primary focus lays at the United Kingdom, having the 2nd most customers for their services. There are currently serious plans to expand to Germany, India and Indonesia.

--Interview concluded.--

David processes the user statistics every week. Every week a newsletter is send through the company with the statistics of each week, compared to last week and the change. At the end of each month, the final statistics are compared to the previous month for a more complete picture. The data for the first five months (January, February, March, April and May) are plotted into figure 4.

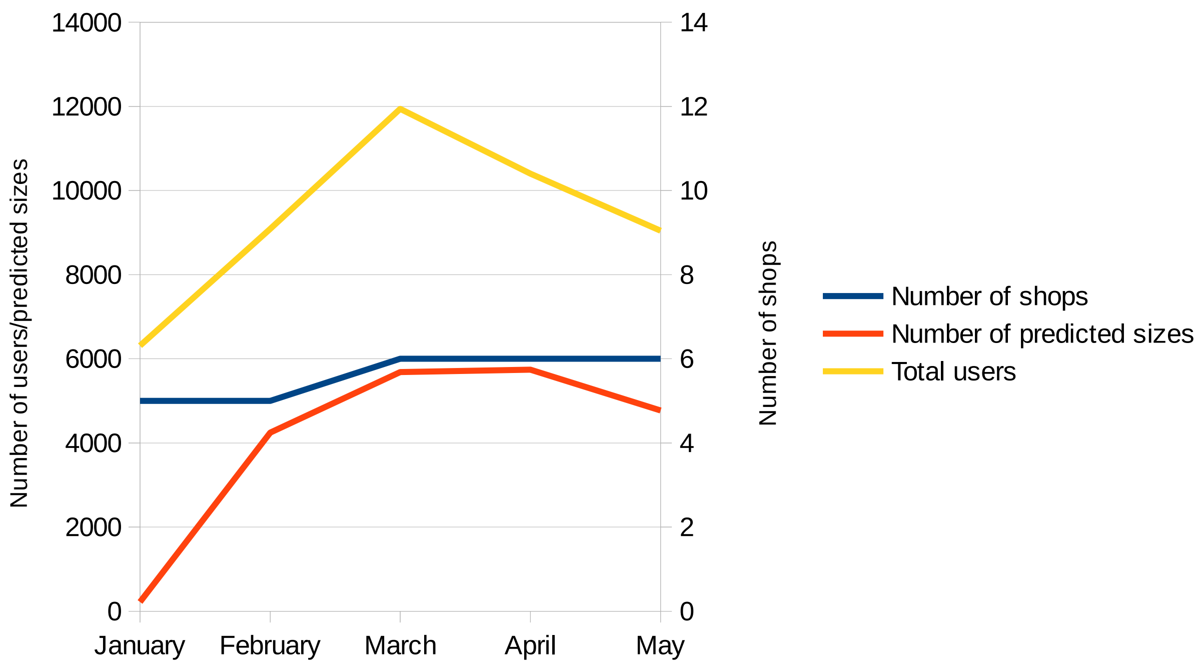


Figure 4: The development of number of shops, predicted sizes and total users.

Please note the number of shops is aligned to the secondary Y axis to make the growth/shrink visible. The company has made a stunning growth in the first month of 2016. The number of predicted sizes has increased with approximately 4000 sizes and the users connecting to the Easysize API (in other words: the customers that are browsing an Easysize enabled web shop) has also increased with approximately 2000.
While the number of total users dropped with almost 2000 in April, the number of predicted sizes remained the same. We do, however see an extra growth in the number of size predictions when a 6th shop was added.

With an extra shop added in May, the load will probably have increased even further. In February were about 849 predicted sizes per shop, in March it were 947 predicted sizes and finally in April we reached an average of 956 predicted sizes per shop. This is an average of 920 sizes per shop. If this trend continues and the company reaches its goal of 15 shops, the number of predicted sizes will be about 13813 per month. If the 2nd goal of 19 new shops in the second half will be reached, we will have a total of 34 shops using the service. This will lead to 31310 predicted sizes on the server per month. The number of predicted sizes may be even larger if one of the shops has a huge customer base.
The number of connecting users will probably be even higher, but fluctuates too much at the moment to get an accurate prediction.

6.2 What is the current and expected load on the service?

The graph in figure 4 shows some interesting data. While the number of shops has not grown or decreased, the number of users and predicted sizes drop. We can think of a number of causes for this, but that is not part of the assessment.

We will use the data used for figure 4 as input for the load calculations. There are 3 calculations made:
1. Average visitors per shop
2. Average number of size calculations made per shop
3. Average calculations per user

These statistics will give us an insight how our web app is used by the web site users. These data will further processed to predict the future of the service, and the load the servers have to deal with. The tables in figure 5, 6 and 7 present the averages on month-to-month base.

	Number of shops	**Total users**	**Average users/shop**
January	5	6315	1263,00
February	5	9091	1818,20
March	6	11944	1990,67
April	6	10399	1733,17
May	6	9046	1507,67

Figure 5: Calculation table for the average number of users per shop

	Number of shops	**Total size calculations**	**Average calculations/shop**
January	5	220	44,00
February	5	4248	849,60
March	6	5685	947,50
April	6	5740	956,67
May	6	4770	795,00

Figure 6: Calculation table for the average size calculations per shop.

	Average users	**Average calculations**	**Average calculations/user**
January	1263,00	44,00	0,0348
February	1818,20	849,60	0,4672
March	1990,67	947,50	0,4759
April	1733,17	956,67	0,5395
May	1507,67	795,00	0,5273

Figure 7: The final results – calculations per user.

Using the same formulas as described earlier in this chapter, while making the obvious changes to the division operations. The new statistic results are:

Average users per shop: (1263,00+1818,20+1990,67+1733,17+1507,67)÷5 ≈ **1662,50**.
Average size calculations per shop: (44.00+849,60+947,50+956,67+795,00)÷5 ≈ **718,60**
Average calculations per user: (0,0348+0,4672+0,4759+0,5395+0,5273)÷5 ≈ **0,50**

With the new values, Easysize would serve an estimate of 11638 customers a month, and with 19 shops an amount of 30658,5 customers will be served in December.
This will lead up to an average of 5030 size calculations in July and 13652 in December.
The data has been lined out in Figure 8.

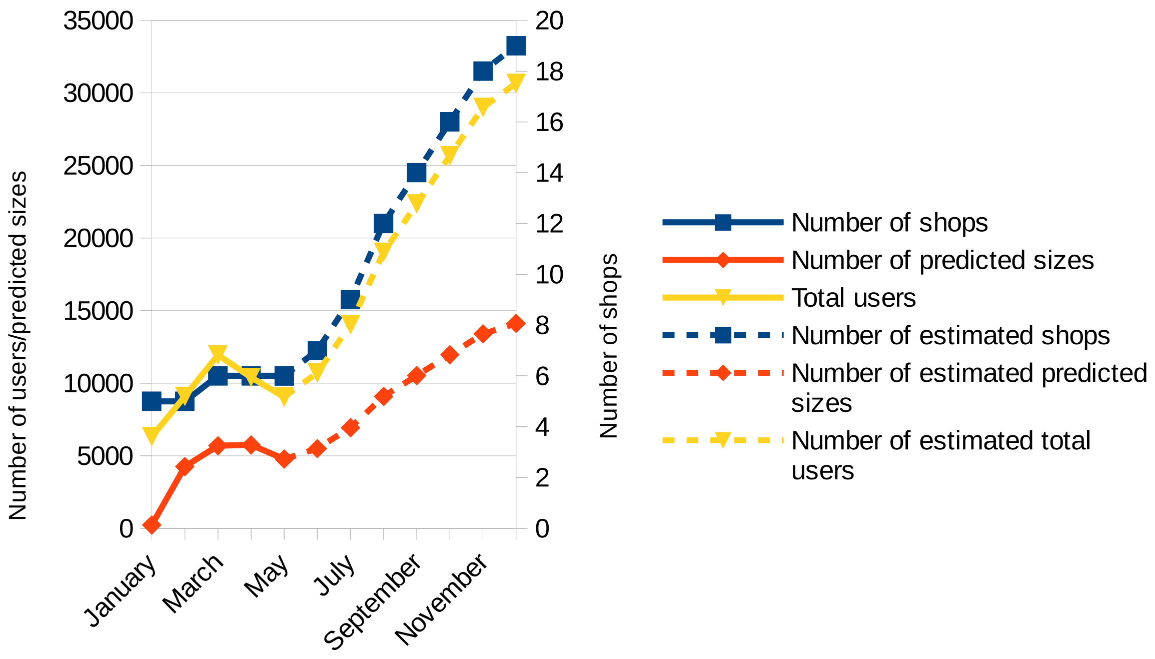


Figure 8: The table with updated values

This seems to be a better representation of the values we are currently experiencing. We expect we will get more users during certain periods in the year, for example the holiday season or start of the spring. This may also explain the peak in users seen in March.

With 30.000 calculations per month in December, it turns out the site gets 40 visitors per hour. 40 users each hour is not very heavy to deal with. However, EasySize is mainly active in Europe, and the most visitors will use the service during the day hours, stressing the servers. As a consequence, the servers are idling at night. This pattern is reinforced by recording in Opbeat: it shows almost no activity at night, but during daytime it can be up to 100 requests per minute.

This estimate is for shoppers only. The load generated by the customer dashboard (for shops) has not been taken into account in these calculations. We are also aware the very low results recorded in January tends to keep the averages down, so I would like to suggest a margin of 1000 visitors per shop in July and 3000 in December. This is the difference when January is left out.
Also, a shop can do better or worse than the average, so it is by no means sure the service will develop as fast or slow as predicted.

6.3 What are the most common causes of slowing down?

There are numerous causes for a server slowing down. This can be broken up into a number of parts. However, most people tend to look at CPU cycles, amount of RAM (Random Access Memory), hard disk space and settings. While the CPU cycles needs an explanation on its own, the RAM and hard disk need to be explained together since they are closely related.

6.3.1 CPU cycles

The CPU is the heart of the computer: It does all calculations. The time the CPU spends on a task is what is actually called load. The article “Understanding Linux CPU load – when should you be worried?” (Andre, 2009, July 31, http://blog.scoutapp.com/articles/2009/07/31/understanding-load-averages) explains nicely about CPU load and what should be optimal, and when CPU begins to congests and slows the server down.

The CPU load explains how much of the CPU time is used. As a rule of thumb, the number of cores is equal to the maximum average load a server. If you have only 1 core, the maximum load is 1.00, 2 cores gives you 2.00 and so on.

If a server has an average load of 1.00, one can say it's the most ideal situation because the CPU is fully utilized. However, if there is a little more calculation power needed, the tasks needs to queue for CPU time, effectively causing a “traffic jam”. One more connection, or a starting planned task, or a system operator logging in via SSH, it may cause the load average to shoot over 1.00 and slow things down. Because there is no headroom, the information from the link above recommends a warning level on 0.30 load average left (on one core it means: warning at load average 0.70).

When the average load is high above the number of available cores, it may indicate a severe problem. Either the hardware is inadequate to handle all the software and its tasks, it could be severely misconfigured, or in the worst case scenario, compromised by a hacker who uses it for unauthorized tasks like mining cryptocurrency or acting as a Command-and-Control server for a bot-net.

6.3.2 RAM and Hard disk

The RAM is special memory the CPU can access immediately. It can retrieve values of it, do some operations on them and store them back in the RAM. Because the RAM can be accessed directly when an application requests it, it makes retrieving the values extremely fast (and making the operations very “cheap” time wise)

The disadvantage of RAM is the information is lost on a shut down (for maintenance or power outage) or reboot (mostly for maintenance). This means it is unsuited for permanent data storage. Hard disks are made for long-term storage, but cannot be directly accessed. When information is needed (like database information) it will be retrieved from the hard disk and put in the RAM. The process of retrieving it from the hard disk is slow, meaning it is rather “expensive” to do it each time.

To save time, the data read are stored in the RAM, in a so-called *cache*. Because they can now be read immediately by the CPU, retrieving is fast and will prevent long wait times. This is the reason when systems often respond slowly to the first requests on the websites, because the caches are empty. This will be solved soon when they get filled with useful data.

Some SQL servers, for example, solve this by implementing a query cache. A page on the Percona blog (https://www.percona.com/blog/2011/04/04/mysql-caching-methods-and-tips/) states MySQL offers such a cache, but a cache who stores database results face a serious consequence: they call it *coarse invalidation.* As soon as data in a table is updated, it will invalidate all data in the cache. This is obvious: the data in the cache does not match its real contents and can no longer be trusted.

Because a cache for information can be a good idea, it not always works as stated in the previous paragraph. A buffer is also in play, especially in SQL operations. The MySQL glossary (https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_buffer_pool) gives a short description of the buffer.
The buffer is meant to protect data: data is copied from the hard disk to the buffer (located in RAM). Now it is in RAM, modifications can be made safely. However, the changed information needs to be written to the disk, a process known as *flushing.* If this were to happen after each update, it would slow down the server terribly. This is because all processes are waiting for the disk, either waiting for confirmation of a successful write, or waiting for the data to be retrieved.

To avoid this, a system uses a method called checkpointing. It records the latest state of the buffer when it was written to the disk. If there are too many changes, the page will be rewritten to the disk. Also, changes will also be written to the disk after a certain amount of time has been passed.

In the book High Performance MySQL, Third Edition the authors Schwartz, Zaitsev and Tkachenko (2012) describe a phenomenon called *furious flushing.* This happens when a server spends too much time writing the new information to the disk, causing the situation described two paragraphs earlier. This behaviour can be configured by a bad flushing algorithm like described in the mentioned book, but also by a misconfigured server (own experience).

The speed of psychical storage (like hard disks) is improving by the continuous research to Solid State Drives (SSD's), who are getting cheaper each day. However, because SSD's yet cannot provide the storage a classic spinning disk can (or they are awfully expensive). Databases are increasingly fitted with SSD's because they offer a speed improvement when retrieving data, but disk access remains an expensive operation. It only mitigates the time a little, but RAM access remains faster.

6.3.3 Network latency

This is a less likely culprit, but also worth mentioning. In a network, the slowest link determines the speed. If it is a slow dial-up connection, one will probably be waiting forever before the page finished loading. A DDoS attack works like that: they just try to clog up the connection lines to the server with so much traffic to cause a traffic jam on the connection leading to the server. While today the server usually goes down because the CPU gets overloaded (see §6.3.1), a small internet bandwidth may be pack it in before the CPU cannot handle it any more.

7. What measures have already been taken to increase the maximum load?

The company was already approaching the limit for their performance goal. The company took some actions, it were the simplest and most obvious ones.
The measures taken are listed here, to narrow our focus by ruling out other potential problems that can slow down the server.

As stated before, the company has three servers for the service. Under the hood they all run the same basic configuration.
The servers are provided with 14 gigabytes of RAM, 4 CPU cores each and 30 gigabyte of hard disk space. They run Ubuntu Linux 15.10, MySQL database and nginx and uWSGI for the web application.

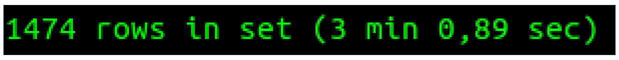
The optimisations already taken were improving the MySQL settings, Python code optimisation and improving the uWSGI settings.
The company was already considering load balancing on multiple servers, however, any of the plans were not yet taken into action.

The list is disappointingly short. Opbeat explained the database was almost always to blame when it came down to performance spikes. Sometimes, an execution time over 30 seconds for a simple query was recorded. This points out to a database problem, which cannot be fixed in code or uWSGI settings. Potential fixes for this problem are explained in Chapter 8.

8. Where are the current and potential bottlenecks and how can they be fixed?

The research from the previous questions point to one main bottleneck at the moment: The database. At some times, it is incredibly slow, taking up to 20 seconds to insert a single record (recorded in Opbeat). Even longer times up to 45 seconds have been noticed on some SELECT commands.

It seems to happen at random times. Combined with the knowledge gained in the previous chapters, it seems the long INSERT or SELECT actions take place when the database is writing data to the disks, or retrieving a lot of data when a certain command is executed. The MySQL server showed it took over 3 minutes to process a complex query, as shown in figure 9.



```
1474 rows in set (3 min 0,89 sec)
```

Figure 9: MySQL shows it needs over 3 minutes to process a query

To get a clear picture and to pinpoint the problem, the system was taking no other load from any unneeded processes. Only MySQL, SSH and required system processes were running. Caches were cold.

From monitoring the server when processing this query, the load on one CPU core was 100% all the time. This is because of MySQL's architecture which does not support multi-threading on a single query: A query can be handled by only one core only. This was further reinforced by doing a second test on a different system.

The first system has two cores from the Intel Xeon E5-2660 type, running on 2.2 GHz. The second test system only got one core of an Intel Core i7-4710MQ, running on 2.5 GHz with automatic over-clock to 3.5 GHz when needed. On multiple tests, the second system was an average of 30 seconds faster than the server.

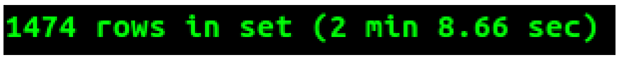
The most obvious conclusion would be to buy servers with a faster CPU. While that probably would indeed mitigate the problem, it is not the most cost-efficient. If a 1000 extra clock cycles per second means an improvement of 30 seconds on one specific query, it means about 5000 extra clock cycles are needed to beat the 30 second line, and those clock cycles are required to run on only one core!
Besides, it still does not explain why some inserts are taking so incredibly long to execute.

8.1 Other database systems

The most possible solution that came up is MySQL is just unsuited for the job. There are a number of other databases available. For the company, the focus was pointed to 2 free alternatives.

The first alternative that came up was MariaDB, one of the newest database systems and started as a fork from MySQL. Their knowledge base (https://mariadb.com/kb/en/mariadb/mariadb-vs-mysql-features/) lists a number of speed improvements in various storage engines and the general query optimizer.

The installation of MariaDB should prove simple according to their own information. After installation, the slow query was re-executed. See figure 10.



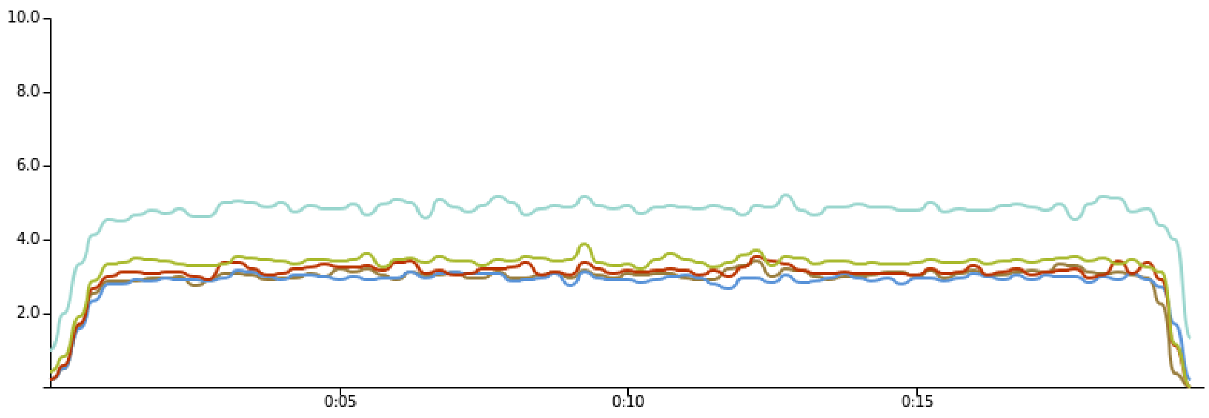
```
1474 rows in set (2 min 8.66 sec)
```

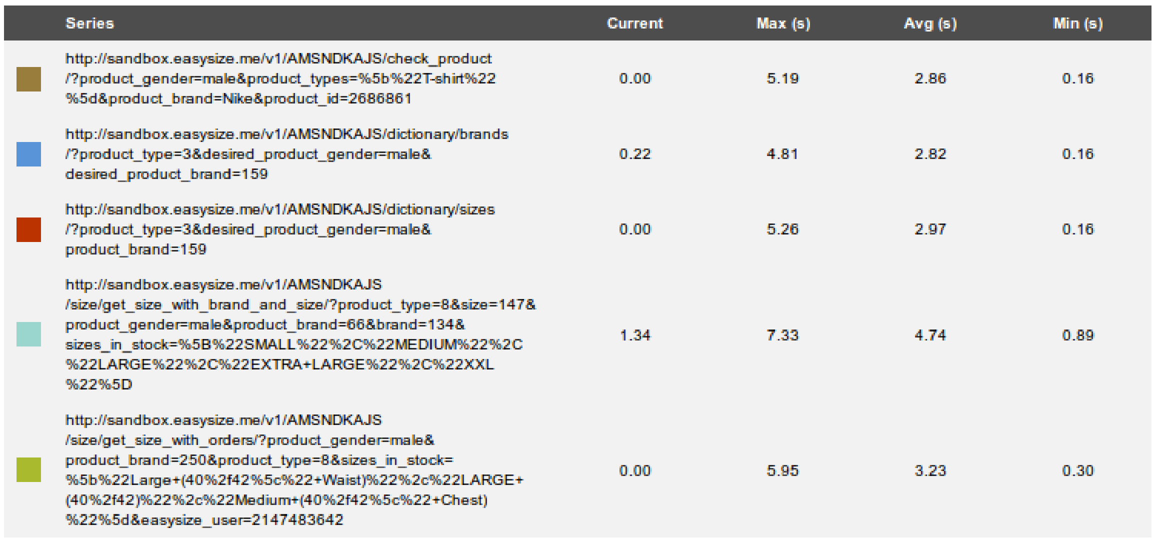
Figure 10: MariaDB finished the query 52 seconds faster

This is only a part of the story. It finishes this particular query quickly, but does not tell anything about the other queries. A more general load testing has to be performed.

For the load testing, the company uses the tool Loadster. The free build-in engine can generate up to 25 simultaneous users. A test runs for 20 minutes, according to the company standards. The test system was restored in its previous state to do a general load test. This provides a general impression of the performance of the database. The spikes may not occur, but also shows how it performs in general.

4.1. Average Response Times by Page



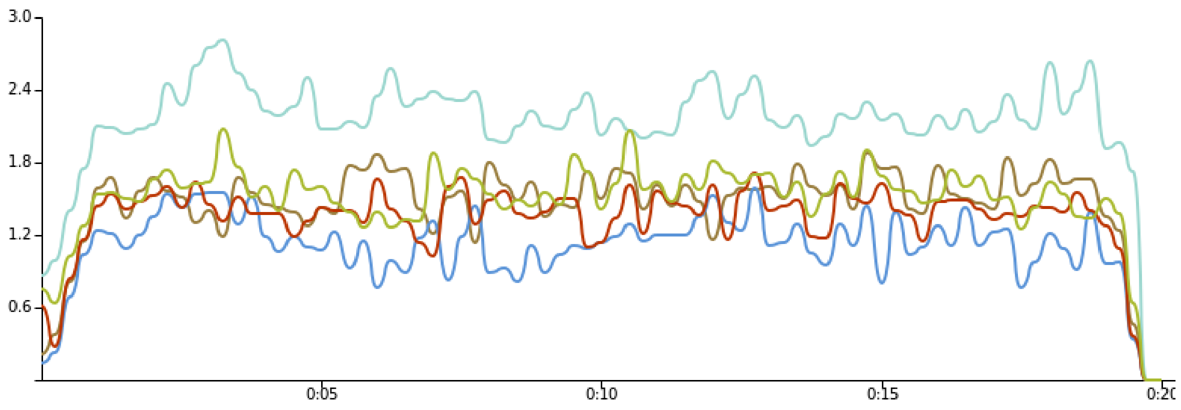


	Series	Current	Max (s)	Avg (s)	Min (s)
🟫	http://sandbox.easysize.me/v1/AMSNDKAJS/check_product/?product_gender=male&product_types=%5b%22T-shirt%22%5d&product_brand=Nike&product_id=2686861	0.00	5.19	2.86	0.16
🟦	http://sandbox.easysize.me/v1/AMSNDKAJS/dictionary/brands/?product_type=3&desired_product_gender=male&desired_product_brand=159	0.22	4.81	2.82	0.16
🟥	http://sandbox.easysize.me/v1/AMSNDKAJS/dictionary/sizes/?product_type=3&desired_product_gender=male&product_brand=159	0.00	5.26	2.97	0.16
🟩	http://sandbox.easysize.me/v1/AMSNDKAJS/size/get_size_with_brand_and_size/?product_type=8&size=147&product_gender=male&product_brand=66&brand=134&sizes_in_stock=%5B%22SMALL%22%2C%22MEDIUM%22%2C%22LARGE%22%2C%22EXTRA+LARGE%22%2C%22XXL%22%5D	1.34	7.33	4.74	0.89
🟨	http://sandbox.easysize.me/v1/AMSNDKAJS/size/get_size_with_orders/?product_gender=male&product_brand=250&product_type=8&sizes_in_stock=%5b%22Large+(40%2f42%5c%22+Waist)%22%2c%22LARGE+(40%2f42)%22%2c%22Medium+(40%2f42%5c%22+Chest)%22%5d&easysize_user=2147483642	0.00	5.95	3.23	0.30

Figure 11: The test results of MySQL

The spike did not occur in this test, however all the methods are performing rather poorly under only 25 users. This needs to be fixed, and MariaDB may be able to do it.

4.1. Average Response Times by Page



	Series	Current	Max (s)	Avg (s)	Min (s)
🟫	http://sandbox.easysize.me/v1/AMSNDKAJS/check_product /?product_gender=male&product_types=%5b%22T-shirt%22 %5d&product_brand=Nike&product_id=2686861	0.00	4.41	1.29	0.11
🟦	http://sandbox.easysize.me/v1/AMSNDKAJS/dictionary/brands /?product_type=3&desired_product_gender=male& desired_product_brand=159	0.00	4.53	1.14	0.08
🟥	http://sandbox.easysize.me/v1/AMSNDKAJS/dictionary/sizes /?product_type=3&desired_product_gender=male& product_brand=159	0.00	6.80	1.22	0.09
🟩	http://sandbox.easysize.me/v1/AMSNDKAJS /size/get_size_with_brand_and_size/?product_type=8&size=147& product_gender=male&product_brand=66&brand=134& sizes_in_stock=%5B%22SMALL%22%2C%22MEDIUM%22%2C %22LARGE%22%2C%22EXTRA+LARGE%22%2C%22XXL %22%5D	0.00	5.75	2.33	0.34
🟨	http://sandbox.easysize.me/v1/AMSNDKAJS /size/get_size_with_orders/?product_gender=male& product_brand=250&product_type=8&sizes_in_stock= %5b%22Large+(40%2f42%5c%22+Waist)%22%2c%22LARGE+ (40%2f42)%22%2c%22Medium+(40%2f42%5c%22+Chest) %22%5d&easysize_user=2147483642	0.00	4.58	1.46	0.16

Figure 11:MariaDB's test result

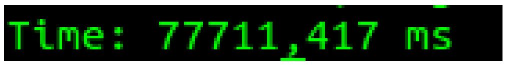
The tests are both impressive and disappointing. The maximum execution time is higher, while the average time is cut in half. The average time is an improvement, but the high maximum times may leave users dissatisfied.

Another option offered on many discussion fora on the internet recommends PostgreSQL. The company has no prior experience with PostgreSQL, and the results on the internet were not easy to verify. The decision was made to test it ourselves to get reliable results.

Moving to PostgreSQL was quite easy. A tool called “pgloader” was able to transfer the data in a reasonable time to MySQL. The documentation (http://pgloader.io/howto/pgloader.1.html) also states the tool works with MS SQL server and mSQL.

The query and the application needed a few tweaks to make it work with PostgreSQL. The raw queries in the application had to be changed to match the PostgreSQL. SQL syntax and conventions. One such convention is working with Strings. Where MySQL uses a double quotation mark (") to start and end a string, PostgreSQL uses only one ('). A GROUP BY clause needs to include the primary key.

The large query was executed first. The results are shown in Figure 12.



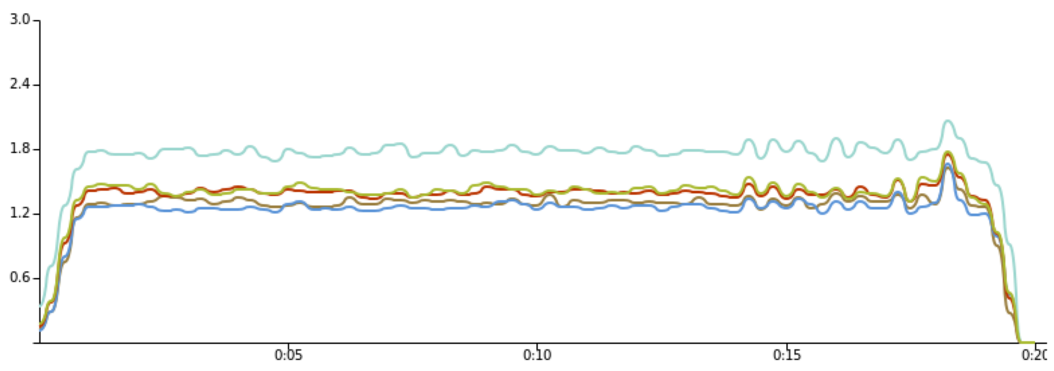
```
Time: 77711,417 ms
```

Figure 12: PostgreSQL just took over a minute to come up with results

This is impressive. Gaining almost 2 minutes when choosing PostgreSQL over MySQL. This is still not the time the company has set to, but shows the solution can partially be found in switching database systems.

PostgreSQL seems to treat other queries just as well. The Loadster test shown in Figure 13 proves it performs on average on par with MariaDB, however the maximum execution time is almost equivalent to the average MySQL execution time.

4.1. Average Response Times by Page



	Series	Current	Max (s)	Avg (s)	Min (s)
🟫	http://sandbox.easysize.me/v1/AMSNDKAJS/check_product/?product_gender=male&product_types=%5b%22T-shirt%22%5d&product_brand=Nike&product_id=2686861	0.00	2.48	1.24	0.11
🟦	http://sandbox.easysize.me/v1/AMSNDKAJS/dictionary/brands/?product_type=3&desired_product_gender=male&desired_product_brand=159	0.00	2.92	1.22	0.09
🟥	http://sandbox.easysize.me/v1/AMSNDKAJS/dictionary/sizes/?product_type=3&desired_product_gender=male&product_brand=159	0.00	3.03	1.34	0.13
🟦	http://sandbox.easysize.me/v1/AMSNDKAJS/size/get_size_with_brand_and_size/?product_type=8&size=147&product_gender=male&product_brand=66&brand=134&sizes_in_stock=%5B%22SMALL%22%2C%22MEDIUM%22%2C%22LARGE%22%2C%22EXTRA+LARGE%22%2C%22XXL%22%5D	0.00	3.53	1.74	0.28
🟩	http://sandbox.easysize.me/v1/AMSNDKAJS/size/get_size_with_orders/?product_gender=male&product_brand=250&product_type=8&sizes_in_stock=%5b%22Large+(40%2f42%5c%22+Waist)%22%2c%22LARGE+(40%2f42)%22%2c%22Medium+(40%2f42%5c%22+Chest)%22%5d&easysize_user=2147483642	0.00	2.88	1.35	0.14

Figure 13: PostgreSQL turns MySQL's average performance into its worst.

This means the high performance spikes shown by MariaDB can be paid off by a slightly higher average response time. The difference is so small the company can afford these few hundreds of seconds extra, to prevent occasional misfires which delays the service up to 6 seconds.

8.2 Too much data

The specifically tested query returns in this specific case over 1400 rows of data, and that number is likely to grow, because it are statistic calculations over a time range. When the company grows the number of users and therefore statistics each month will increase, increasing the statistics into the result the server returns.

Many database systems have the ability to execute certain tasks at a given time. In MariaDB (https://mariadb.com/kb/en/mariadb/events/) and MySQL (https://dev.mysql.com/doc/refman/5.7/en/events-configuration.html) it is known as the Event Planner or Event Scheduler.
However, PostgreSQL seems to not have such a scheduler, but there seems to be tools available who can take over this job.

The event planner offers a nice ability. When executed once a day during the quiet hours of the night, the query can be run once and cache its result. The results can then be retrieved direct from the memory and should be ready within the set deadlines.

These slow, large statistics queries started to “seriously threaten our scalability” (personal communication). When the event planner ability was investigated, an interim advice was issued to retrieve and cache the results of these queries during night. This would solve the imminent problems facing the company.

To demonstrate the event planners' ability, a method was made to copy the contents of a large table to a temporary table, residing only in the server's memory. When this test succeeds, the decision was made to specialize this solution further and cache the results of the statistic queries once a day during night.

The consequence was the results showed the statistics up to last day. The real-time statistics (from up to the last minute) was something the company was willing to sacrifice. They justified it by taking Google Analytics as example, who does not show real time statistics, only from a short while ago.

The result was exactly what the company was looking for. After the page was updated to use the cached results instead and refreshing them once a day at night, the page was ready within the set deadline. This was a huge improvement that was released in the customer version a few days later.

8.3 Conclusions (summary)

The problem turned out to be spit into two parts: On one hand, MySQL seems to be unsuited for the needs of the company. On the other hand, there is too much data to process reliably in a timely manner. The scalability on the current hardware (without buying extra servers) can be improved to switch from MySQL to PostgreSQL, which performs better, also on the small size calculation queries.
The queries used on the statistics dashboard returned a lot of data and cannot be processed in time. The data needs to be retrieved on a set time (during night) and cached to use during the day. It can

be refreshed each night with a scheduled event.

To address the imminent scaling problems the company is facing with the dashboard for customers, the caching tactics described in §8.2 has been implemented. However, the company still uses MySQL for data storage.

9. Main answer

With all the sub questions answered, the main question can now be answered. The main question is

How can the service and its infrastructure be improved to maintain the performance requirements?

The answer is not a simple sentence of one acclimatization. It is a number of things that needs to be done to make sure the company can keep up with demand, now and in the future. In December, they can expect about 30.000 users that month (see §6.2). While that are 40 users each hour, they will visit the shops during daytime and make several requests to the service. Gulnaz Khusainova discovered through analysis customers “like it to play around with EasySize [sizing tool]” (personal communication). This means even more requests to the service, so the extra requests needs to be taken into account.

The first thing to do this is to move the database to its own server, separating the Database Tier from the Web/App tier. This will give the database extra resources to process calculation-intense queries faster. This also ensures scalability if they want to implement load balancing one day.

For the slowest queries that require lots of data that does not rapidly change (like statistic data) can be retrieved once a day and cached for fast access. Especially the customer dashboard will benefit from this, and has already been implemented.

During the last months, the company is growing rapidly and is expanding to East Asia. The statistics collecting during night can harass the users from Asia by putting their requests into a queue that will last several minutes. Load balancing can be used here, by having 2 database servers who collect this data one after another. For example; the first server collects it at 3 o'clock AM (GMT), the second does this at 3 o'clock PM (GMT). This may keep the service responsive, since there is one database system ready for use.

The last thing the company can do to fix this problem is to switch to PostgreSQL. It shows it performs much better than MySQL and on par with MariaDB, with some added benefits. To further reinforce this solution, an advice report has been written and can be found in the attachments.

Sharding and partition were also considered, but blocked up on each own in its own way. First, the company uses MySQL's InnoDB Storage Engine for its tables. While InnoDB supports table partitioning according to the documentation (https://dev.mysql.com/doc/refman/5.7/en/partitioning-overview.html), another page (http://dev.mysql.com/doc/refman/5.7/en/partitioning-limitations.html) warns it cannot enforce foreign keys, threatening data integrity.
Sharding was also mentioned, which is effectively partitioning a table and put each partition on its own server. It is clearly depicted on one of MySQL's product pages (https://www.mysql.com/products/cluster/scalability.html). At the moment, this approach is a too large solution for where the company is right now regarding size and server load. Implementing sharding right now would mean extra cost to buy extra servers, and it would be like using a sledgehammer to crack a nut.

10. Delivered products

At the beginning of this assignment, a workable proof-of-concept was part of the agreement. However, during the investigation, an advisory was also created to reinforce the decisions made, and to prove which solutions were best and why. This results in a report as well as a proof-of-concept.

10.1 Advisory

From extensive research trough database systems, three systems were compared to each other. The chosen systems were MySQL, because that is what the company currently uses. The competitors were MariaDB and PostgreSQL. From testing them each in turn on a test system PostgreSQL came out as a clear winner, MariaDB came in second but profits seriously from its MySQL heritage, and surprisingly MySQL came in dead last, performing worse than the other 2 systems on all aspects.

The advice from the report is to switch to PostgreSQL, but consider the consequences. The biggest one is the absence of a build-in event planner, but other tools to achieve this exist. The second disadvantage involves the raw queries in the application. They need to be changed to PostgreSQL's SQL syntax, which differs slightly from MySQL's syntax.
Therefore, MariaDB is given as an option to consider. The user simulation (load) test was comparable to PostgreSQL, but shows some high spikes. The statistics query runs slower, taking up to 2 minutes, but can be solved using the caching technique already in place.

The advice clearly advises PostgreSQL above MariaDB, but if the consequences are considered too big, MariaDB can take over the job from MySQL, but other measures like load balancing or partitioning (which can be problematic) will have to be implemented sooner.

10.2 Proof-of-Concept

The proof-of-concept was made before the advisory, since it was used to test the improvements and to tweak the configuration of the tested databases. The default settings almost never turn out to be the best. After tweaking the settings, the load test was done, and a separate benchmark on the database was also performed to see how it would run with optimal settings.

The finally delivered product is a testing server containing the company's database imported in PostgreSQL, with the raw queries in the source code updated to PostgreSQL. The developers can use the version control system to re-check the code for missed spots and easily implement it into their production base.

11. Results vs. Plan of Approach

The tested results now have to be tested against the Plan of Approach who was written at the beginning of this assessment. A number of 6 quality criteria have been defined to test the delivered products to the goals set at the beginning.

11.1 Sources

The Plan of Approach states “the sources used in the report have to be reliable”. The majority of the sources used in the report is based on official documentation, considered almost the best quality of source available. It is often updated to the latest version, and older information from previous versions can still be retrieved. A number of private websites or blogs was also used, but were mostly an exception to complement information the documentation was lacking once in a while. With this strategy this requirement can considered **satisfied.**

11.2 Scalability

The company only has a limited number of options. The choice to switch databases will improve further scalability, but only on the current hardware. Load balancing or a similar technology will have to be implemented one day, but was not taken into account, since the company only has a limited supply of resources.

The used databases are simple to implement and especially PostgreSQL can take quite some more load before it starts to break up. Since the database systems itself are free, the company will not lose too much money when customer numbers drop, which is possible but does not seem likely at the moment. At this point, upgrading to MariaDB is almost free even time-wise because it's incredible compatibility with MySQL.

For now the imminent problems are solved, and this can be seen as **satisfied.**

11.3 Considerable gain

This one is very clear; the whole investigation was focussed around this. PostgreSQL usually cuts execution time in half, sometimes even more than that. This one was our main point, and the results are what was requested. This point is without a doubt **satisfied**.

11.4 Clear implementation instructions

For MariaDB, there is not much to be done (while it also depends on the MySQL version). Installing is very easy and can be done within an hour. After that, the service can continue its normal operation. The application itself does not have to be changed. The advisory contains a special chapter how to do the upgrade safe.

For PostgreSQL, the advisory also provides how to migrate the database and update the application to work with PostgreSQL. It covers moving the tables using the 3rd party tool “pgloader” A script

for this program is provided in the report. It also explains how the raw queries have to be updated to continue to work with PostgreSQL. For example: A GROUP BY clause has to include the primary key, and an IN clause has a different syntax compared to MySQL.
The testing done to see if the code was still correct returned the expected, correct results.

Having a guide by hand to implement the recommended changes and scripts provided to make it even easier, this requirement is **satisfied**.

11.5 Language

The set language for the delivered documents was set to British English. EasySize consists of employees from various countries, and English is set as main language for communication. This requirement is clearly **satisfied.**

11.6 Literature types

The biggest source of information was the internet for the official documentation of the used programs. Also the internal design documents were read, but provided not too much information, so the employees were frequently asked questions to clarify things. The meetings we had occasionally also revealed some information about the systems and problems.

However, no books about load balancing have been read. This was promised in the Plan of Approach, but load balancing turned out to be quite expensive for the company, so the decision was made to try to improve the response time by using already existing hardware by tweaking the database.
It turned out it was better to try out some other systems as well.

The only book that has been partially read was *High Performance MySQL, Third Edition*. It makes no sense to read the whole thing, but only the interesting parts. Useful chapters included indexes, benchmarking and caching.

Because of the lack of real books and no load balancing, the fulfilment of this requirement is **weak.**

12. Evaluation procedure

12.1 Meeting the company

The procedures used in the company are iterative. Each day starts with a “daily stand-up”, comparable to the daily SCRUM meeting. At the end of each week a meeting was organized with the developers, so everybody knows what has been done the past week and what tasks are still open. This will provide a steady course of the company instead of risk everybody going loose, causing new issues and also causes issues not to be fixed.

To assist in these tasks, the company uses the web application Trello, which works with “cards” to describe tasks. A deadline can be set on each card and assigned to one or multiple people. The cards are put on different boards. This board tactic helps in tracking the progress of a card. When a card is finished, it will be archived: it is still retrievable, but not automatically visible on one of the boards.

The goal what the company wanted to achieve became clear during the first psychical job interview. To know more about the systems an interview with the developers was done and the documentation was read. This provided a simple idea about the company

12.2 Plan of Approach

The plan of approach was a signpost in the whole investigation. It shows the quality requirements set, the deadlines to be met and reminds of the goals the company wants to archive. In this case, it also guarded the quality of the used sources by setting requirements to them. The document helps to think objective but critical about sources. This revealed some interesting information regarding the sources. Sometimes, a blog post that just looks like any other turns out to be written by a specialized database performance expert, making it worthwhile to read it again and test more of the recommendations in it.

12.3 Communication

The workplace in the company was set next to David Babayan, so questions could be easily asked. Also, a meeting was organized each week in the beginning to make up the current situation. Later it was less needed because of the ongoing research. These meetings were highly appreciated because it was a source of new ideas, and also a feedback of the progress that was made.

The communication with the supervising lecturer worked, but was not ideal. It took a while before a phone link was established. He explained he does have a company phone and an associated number, but does not use it and offered to use his private number instead. After his minor bump I managed to reach him almost always when he was needed, but took long time to check the concept versions and plan of approach documents. Sometimes a call was required.

12.4 Planning

The Plan of Approach provided a crude planning, which bit back in the end. Things were sometimes done on the last opportunity and there were a few days work laid still because of unexpected problems. With the timetable set it provided not enough information about the phrases and iterations. This should really be done better next time. It will almost surely improve efficiency as well.

12.4 Finding the bottlenecks

To solve the problem, the source of the problems needs to be found. The measures that were already taken were recorded, so they could be verified quickly. This also prevents re-inventing the wheel only to discover it has already been done.

The database turned quickly out to be the biggest bottleneck. However this is only part of the story. The source of the slow database needs to be identified to do something about it. Chances are it is just misconfigured and killing performance in that way. Another possibility are tables with bad (or no!) indexes.

However, after an extensive research the database configuration was just fine. Indexes did exist on tables; however one table had an unused index. The queries are mostly generated by an ORM and cannot be changed. The raw queries in the application turned out to be a solution to work around some of the limitations of the used ORM and were an exception.

The book *High Performance MySQL, Third Edition* was of great help to identify possible solutions regarding indexes and benchmarking. This lead to extensive benchmarking of the MySQL, MariaDB and PostgreSQL servers. These results are included in this document.

Also percona came up with number of MySQL solutions; however none of them really worked. It only gained tenth of seconds or had other huge consequences. It is interestingly to read some of the pages though, because it also explains how MySQL works “under the hood”.

Unfortunately, much time was lost on a solution called memcached. Memcached is an “active cache”, which means it will fetch the results if it is not yet in the cache. An attempt was made to use it as a query cache, because MySQL's own query cache caused high performance spikes every 7~10 minutes. This solution uses NoSQL however, and it would be hard to connect the application to memcached and correctly interpret the results.

This caused for a block, leaving nowhere to look. However, the remark of Emilis Sapronas that he was willing to experiment with other systems (he mentioned FreeBSD as an alternative operating system) rose awareness of the fact the database system is not put into concrete. A research to other systems began the very same day, and a few days later 2 main candidates were selected: MariaDB and PostgreSQL.

From that moment, the research got a boost and the momentum was used to seriously test them. The tests also broke the testing server twice, but since it is a virtual machine, its state right after installation was saved, so replacing it took about twenty minutes.

The system breakdowns were a harsh lesson what to do and what to avoid during a database switch. This was very valuable information for the company. These mistakes can now be avoided and are recorded in the advisory.

The final verdict explained that PostgreSQL and MariaDB's were both better than MySQL, where PostgreSQL won with a slight margin regarding maximum (worst) performance and its performance on calculation-intensive statistics queries.

However, the block described in the middle of this section also caused a delay for the investigation that resulted into time pressure at the end of the internship. This is something that should really have been coordinated better. Maybe David Babayan should have been asked for advice.

12.5 One last question…

After the discovery MySQL was the worst performer of all solutions, the question arose why the choice for MySQL was made in the first place. It could have nullified the recommendations, but the reason for using MySQL was just as surprising as disappointing.

Both developers, Emilis Sapronas and David Babayan stated “it [MySQL] was just there” (personal communication) when they joined the company. Emilis Sapronas was the company's first developer, so it must be Gulnaz Khusainova who made the decision. She stated it she choose for MySQL because it was (more or less) a standard solution.

Now while it is now clear MySQL is not a hard requirement, but chosen as a standard option, the path to migration should be paved. In the meantime, the company is expanding rapidly and with this, a potential great future lies ahead of EasySize.

13. References

Andre. (2009, July 31), Understanding Linux CPU load – when should you be worried? [Blog post] Retrieved from http://blog.scoutapp.com/articles/2009/07/31/understanding-load-averages

Django Software Foundation (2015), *Databases* [Official documentation]. Retrieved from https://docs.djangoproject.com/en/1.8/ref/databases/

Django Software Foundation (2015), *Django appears to be a MVC framework, but you call the Controller the "view", and the View the "template". How come you don't use the standard names?* [Official FAQ]. Retrieved from https://docs.djangoproject.com/en/1.8/faq/general/#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names

MariaDB Foundation (n.d.), *MariaDB versus MySQL – Features.* Retrieved from https://mariadb.com/kb/en/mariadb/mariadb-vs-mysql-features/

MariaDB Foundation (n.d.), *Events overview* [Official documentation]. Retrieved from https://mariadb.com/kb/en/mariadb/events/

MySQL (n.d.), *Event scheduler configuration* [Official documentation]. Retrieved from https://dev.mysql.com/doc/refman/5.7/en/events-configuration.html

MySQL (n.d.), *MySQL Glossary.* Retrieved from https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_buffer_pool

MySQL (n.d.) *MySQL Cluster: Scalability*. Retrieved from https://www.mysql.com/products/cluster/scalability.html

MySQL (n.d.) *Overview of Partitioning in MySQL* [Official Documentation]. Retrieved from https://dev.mysql.com/doc/refman/5.7/en/partitioning-overview.html

MySQL (n.d.), *Restrictions and Limitation on Partitioning* [Official documentation]. Retrieved from http://dev.mysql.com/doc/refman/5.7/en/partitioning-limitations.html

Nginx, Inc. (n.d.), *Load Balancing with Nginx and Nginx Plus, Part 1* [Official documentation]. Retrieved from https://www.Nginx.com/blog/load-balancing-with-Nginx-plus/

Nginx, Inc. (n.d.), *Load Balancing with Nginx and Nginx Plus, Part 2* [Official documentation]. Retrieved from https://www.Nginx.com/blog/load-balancing-with-Nginx-plus-part2/

Nginx Inc. (n.d.), *nginx* [Official documentation]. Retrieved from http://nginx.org/en/

pgloader (n.d.), *pgloader(1) – PostgreSQL data loader* [Official documentation]. Retrieved from http://pgloader.io/howto/pgloader.1.html

Poort, F (2014, March 31st), *Twee dagen uitstel bij belastingaangifte vanwege storing* [News article]. Retrieved from http://www.nu.nl/algemeen/3740573/twee-dagen-uitstel-bij-belastingaangifte-vanwege-storing.html

Read the Docs, uWSGI (2014), *The uWSGI project* [Official documentation]. Retrieved from https://uwsgi-docs.readthedocs.io/en/latest/

Reitz, Kenneth (n.d.), *Installing Python on Mac OS* X. Retrieved from http://docs.python-guide.org/en/latest/starting/install/osx/

Swanhart, J. (2011, April 4), MySQL caching methods and tips [Blog post]. Retrieved from https://www.percona.com/blog/2011/04/04/mysql-caching-methods-and-tips/