Geographically-aware scaling for real-time persistent websocket applications.

Master's Project in Software Engineering

Lukasz Harezlak
lukasz.harezlak@gmail.com

Summer 2015, 51 pages

Supervisor: Tijs van der Storm
Host organisation: Instamrkt, https://instamrkt.com

Universiteit van Amsterdam
Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Master Software Engineering
http://www.software-engineering-amsterdam.nl

Contents

Abstract

This section summarises the content of the thesis for potential readers who do not have time to read it whole, or for those undecided whether to read it at all. Sum up the following aspects:

- relevance and motivation for the research
- research question(s) and a brief description of the research method
- results, contributions and conclusions

Kent Beck [JBB+93] proposes to have four sentences in a good abstract:

1. The first states the problem.
2. The second states why the problem is a problem.
3. The third is the startling sentence.
4. The fourth states the implication of the startling sentence.

Chapter 1

Introduction

"Does this solution scale?"
— Every business person ever

Everyone these days, probably more than ever, wants to build solutions that scale. Together with the globalization of our lifes and businesses grows the need for global, distributed, scalable software systems.

To address this needs, more and more providers offer Cloud Computing solutions 2.1.5, giving rise to the new architecture model called *Infrastructure as a Service (IaaS)*. This makes building scalable software systems easier, since the system engineers do not have to design and maintain their proprietary server infrastructure anymore. This also allows to build systems that are more efficient in terms of resource usage - the existing *IaaS* solutions offer scaling up and down capabilities in response to changes in demand for system's services.

As more and more systems are deployed to the cloud, a need for measuring their properties becomes more and more ubiquitous.

In order to reason about scalablity-related properties of the system, one needs to define this notion considered by many as vague. For reasoning about cloud scalability, I find the following definition most relevant:

> [WG06] Scalability is an ability of a system to handle increased workload by repeatedly applying a cost-effective strategy for extending a systems capacity.

Furthermore, to have a scientific discussion about scalability, quantifiable models and metrics are necessary. Selection of an appropriate scalabality measurement model was a big part of my initial study.

1.1 Initial Study

Initial study pointed me to the relevant scalability vectors discussed by researchers and engineers regarding cloud computing. Each of them has a dedicated section in the Background Chapter 2.1:

- General Cloud Scalability (2.1.5)
- Data Layer Scalability (2.1.6)
- WebSocket Scalability (2.1.7)

The study yielded multipile scalability measurement models. I analyzed the work of Jogalekar and Woodside [JW00], Srinivas and Janakiram [SJ05], Cooper et al. [CSE+10] and multiple others.

Some of the proposed models are note applicable since they were designed for different types of scalability - not for a cloud-based distributed system, but for instance for multiprocessor or multithreading scalabilities.

The models I selected to use for measurements in this research were proposed by Pattabhiraman et al. [GPBT11]. They were designed specifically for the task I am performing - measuring properties of a distributed system deployed in a cloud environment. Their models are also customizable - one can plug in metrics one deems relevant, which allowed us to select metrics that are representative of the business goal of the System Under Test 4.

Selection and application of literature-based models is described in full in Scalability Measurements chapter 6.

1.2 Problem Statement

The problem that I study in this research concerns application stack decomposition of the System Under Test 4. It can be generalized to scalability of systems with similar characteristics:

- utilizing real-time, uncacheable data,
- utilizing user-generated data,
- broadcast to user base distributed globally,
- broadcast using the WebSocket protocol 2.3,
- deployed in the cloud,
- dynamically scalable (up and down).

This regards the size and geographical placement of WebSocket server instances, database and cache server instances (with data distribution), routing between them and iternal layer communication.

1.2.1 Research Questions

1. What's the architecture decomposition stack that:
   (a) guarantees data delivery to geographically distributed users with minimal, consistent and manageable latency?
   (b) scales up and down most effectively in response to demand changes? (This provides an additional challenge for applications using the WebSocket protocol, since in order to stop an instance one needs to make sure it does not serve any sessions; stateful connections need to be terminated [GB14].)
   (c) guarantees lowest cost and best degree of utilization of the employed resources within a data center?
2. Does architecture with geographical awareness of its users' distribution provide better Quality of Service than the baseline architecture?

Answers to these questions are evaluated using the models and a set of metrics described in detail in a dedicated chapter 6.

1.2.2 Solution Outline

In order to answer the Research Questions, a solution consisting of the following steps was developed:

1. `Preparing the load test suite`. It is used to simulate geographical distribution of users and load they generate as well as to collect latency-related metrics of WebSocket handshakes and message delivery. Described in details in a Load Testing Framework section 5.3.
2. `Preparing the load test suite`. It is used to collect data from the application servers. This consists of collecting Amazon's CloudWatch[1] metrics and our custom metrics (e.g. process memory usage). Metric collection process is described in details in a dedicated section 6.2.1.

[1] `http://aws.amazon.com/cloudwatch/`

3. `Feeding the collected metric into scalability measurement models.`

4. `Automatic scalability management`. This is performed using Amazon's CloudWatch Alarms[2] triggering Auto Scaling Groups [3] capacity changes. In order to include our custom metrics in orchestrating the scalability process, a module based on Amazon's Python library (Boto[4]) was built.

1.2.3 Research Method

Software Engineering is a relatively difficult field to investigate. Most of the problems are of design (rather than pure scientific) nature. West Churchman coined a specific term for these kind of problems [Chu67]. He calls them `wicked`, since they are resistant to resolutions.

Easterbrook et al. [ESSD08] claim it is often difficult to identify the true underlying nature of the research problem and thus the best method to research it. In their work, they name and compare five most common classes of research methods to select from:

- controlled experiments,
- case studies,
- survey research,
- ethnographies,
- action research.

They help to select the method by first establishing the type of research question being asked:

- existence question - *Does X exist?*,
- description and classification question - *What is X like?*,
- descriptive-comparative question - *How does X differ from Y?*.

The questions of this research are of the last type. The authors [ESSD08] suggest pinpointing up-front what will be accepted as a valid answer to the research question. I answer this question in a separate section 6.1.

The detailed description of each of the methods helped me to settle for the `controlled experiment` as a research method for this research. It is well-suited for testing a hypothesis where manipulating independent variables has an effect of dependent ones, which is exactly the case of my research. The manipulated variable is architecture decomposition, and the measured ones are determined by scalability measurement models described in Scalability Measurements 6 chapter.

1.2.4 Research Difficulty

The experiments is performed in a shared cloud environment, which is inherently unpredictable and changing. Non-deterministic network conditions play a significant role here. Aside from that, one has to acknowledge the fact that control over hardware in a cloud environment is by definition given up by the researcher.

Designing and executing a test, yielding statistically relevant results, where all the necessary variables are controlled (within reasonable boundaries) is a huge challenge. Bornholt [Bor14] mentions that even the linking order in the process of generating binaries can have an effect of the performance of the benchmarked system!

The focus of the project is on scaling a websocket application. This is a relatively new technology, thus finding proper scientific coverage is not trivial.

[2] http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/AlarmThatSendsEmail.html
[3] http://aws.amazon.com/autoscaling/
[4] http://aws.amazon.com/sdk-for-python/

Some of the necessary development and data collection tools are relatively low-level and require thorough understanding of UNIX operating systems, on which the servers are deployed. The same is applicable to tuning the operating system for optimal TCP-related performance 5.4.

Simluating the geo-location of clients with reasonable accuracy also proves problematic.

Overall, its a complex project requiring knowledge of multiple aspects of both hardware (TCP, WebSocket protocol, networks, operating system), software (architecture, cloud computing, data replication/sharding).

1.2.5 Hypothesis

The correct geographical decomposition of the stack of The System Under Test 4 can lead to improved performance in comparison with the Baseline Architecure 5.2.

1.3 Contributions

The research will result in the following contributions:

1. Architecture decomposition analysis for system's meeting the criteria described in the Problem Statement section 1.2.
2. Case Study on Amazon Web Services[5] and usability of their cloud stack for applications using WebSocket protocol.
3. Open sourced pieces of code for:
   - metrics collection (both from the load test suite and the servers),
   - auto-scaling execution,
   - constructing scalability models and populating them with collected metrics,
   - remote execution of load tests on a fleet of client-simulating servers.

   These are described in details in Experiment Deliverables section 5.10.

1.4 Related Work

There is two key papers this research is based on. First of them is *Multi-Cloud Provisioning and Load Distribution for Three-Tier Applications* by Grozev and Buyya [GB14]. The second one is *SaaS performance and scalability evaluation in clouds* by Pattabhiraman et al [GPBT11].

Multi-Cloud Provisioning and Load Distribution for Three-Tier Applications

In this work, the authors tackle the problem of deploying applications across multiple clouds while satisfying sytem's nonfunctional requirements (in their understanding, application deployed in Amazon's *us-east-1* region and *eu-west-1* region would be deployed to multiple clouds). They propose algorithms for resource provising and load distribution. They focus on resource utilization and cost and legal regulations regarding serving users and servers geographical location and include them in their algorithms.

In order to satisfy the requirements, the authors had to develop their own routing stack. Both Amazon's services involved in the routing process – Route 53 [6] and Elastic Load Balancing[7] – do not consider application's regulatory requirements when selecting a data center site. Moreover, they do not consider the cost and degree of utilization of the employed resources within a data center.

This is how they perform cloud selection to serve a user:

[5] http://aws.amazon.com/
[6] http://aws.amazon.com/route53/
[7] http://aws.amazon.com/elasticloadbalancing/

1. As a first step, the user authenticates with one of their entry point servers. At this point, the entry point has the users identity and geographical location (extracted from the IP address).

2. As a second step, the entry point broadcasts the users identifier to the admission controllers of all data centers. They call this step *matchmaking broadcast*.

3. The admission controllers respond to the entry point whether the users data are present and if they are allowed (in terms of legislation and regulations) to serve the user. In the response, they also include information about costs within the data center.

4. Based on the admission controllers responses, the entry point selects the data center to serve the user and redirects him or her to the load balancer deployed within it. The entry point filters all clouds that have the users data and are eligible to serve him or her. If there is more than one such cloud, the entry point selects the most suitable with respect to network latency and pricing. If no cloud meets the data location and legislative requirements, the user is denied service.

5. After a data center is selected, the user is served by the Application Server and Database servers within the chosen cloud.

Below I list most relevant takeaways from their work.

Sticky Load Balancing - it is a technique of load balancing that routes same users to the same backend server instaces. They utilize it in their work - I cannot in this research. Amazon's stack does not offer sticky load balancing for protocols other than HTTP.

Cloud Server Instance Termination - it is not beneficial to terminate a running VM ahead of its next billing time. It is better to keep it running until its billing time in order to reuse it if resources are needed again.

Cost Minimization - this is not an automated feature cloud providers offer. Authors advice on how to include it in a cloud controlling stack. It is an important business goal in my researh.

Latency Minimization and Approximation - authors, just as myself, treat latency as one of the key objectives in the process of optimizing their cloud stack. They approximate latencies and users geographic location using tools PingER[8] and GeoLite[9]. PingER is used to approximate latencies between user and a cloud and GeoLite to map users IP addresses to geographical coordinates. In my research, I do not need to use tools like that since Amazon's stack offers this in sufficient capacity.

Why is this work relevant? The authors are attempting to solve the same problem - how to distribute and route users between multiple clouds given arbitrary conditions. They work with a similar technology stack as I do.

SaaS performance and scalability evaluation in clouds

Pattabhiraman et al realized that cloud computing and its measurement provides a new set of challenges when it comes to measuring performance, as opposed to measuring performance of traditional software systems. They list key points for measuring cloud applications - validating and ensuring the elasticity of scalability and evaluating utility service billings and pricing models. Both are important in my case. They raise an interesting question regarding the latter:

> How to use a transparent approach to monitor and evaluate the correctness of the utility bill based on a posted price model during system performance evaluation and scalability measurement?

[8] http://www-iepm.slac.stanford.edu/pinger/
[9] http://dev.maxmind.com/geoip/legacy/geolite/

The authors divide the performance indicators into three groups and propose formal quantifiable models for each of them:

1. computing resource indicators (CPU, disk, memory, networks, etc.),
   - Computing Resource Allocation Meter (CRAM),
   - Computing Resource Utilization Meter (CRUM),
2. workload indicators (connected users, throughput and latency, etc.),
   - System Load Meter (SLM),
3. performance indicators (processing speed, system reliability, availability)
   - System Performance Meter (SPM).

They also propose model which allow to combine the above: System Capacity Meter (SCM), System Effective Capacity Meter (SEC), Effective System Scalability (ESS), and Effective Scalable Range (ESR). Values that are fed into these models are customizable. The models are described in detail in the Scalability Measurements chapter 6.

Authors perform a case study with Amazon's AWS stack. The case study consists of performed tests and measurements taken on scaling up and scaling down their application. It provides some warnings for me to consider:

- Cloud limitations need to be taken into account. For instance, memory usage data is not collectible from their server instances using their services.
- One needs to be aware of hidden costs that cloud providers excel at introducing.
- One needs pay attention to inconsistencies in performance and scalability data, especially data collected using third-party tools and.

More related work is described in the following chapter - Background 2.

1.5 Document Outline

In this section I outline the structure of the thesis - all chapters with short summaries.

1. `Background` - contains a short summary of the literature study. Describes researched work in the relevant fields 2.
2. `Benchmarking` - contains a description and guidelines regarding the benchmarking process, which constitutes a big part of this research. Lists typical benchmarking mistakes and what I do to avoid them 3.
3. `The System Under Test` - contains a description and guidelines regarding the benchmarking process, which constitutes a big part of this research. Lists typical benchmarking mistakes and what I do to avoid them 4.
4. `Experiment Outline` - contains a description of the conducted experiment, its goals and deliverables. Full description of the compared architectures and cloud stack on which the research is performed is also included in this chapter 5.
5. `Scalability Measurements` - contains a description of literature-based scalability metrics and models. Lists the models and metrics selected for measurements in this research, as well as all metrics collectible from the AWS cloud environment 6
6. `Experiment results` - contains the data collected in the experiments 7.
7. `Evaluation` - contains evaluation of the experiment and analysis of the results 8.

8. `Conclusions` - contains a list of conclusions derived from the analysis of the experiment results 9.

9. `Further Work` - contains a description of relevant, testabl variables I did not have enough time to cover in this research. 10

Chapter 2

Background

2.1 Scalability

Scalability seems to be a notion that everyone intuitively grasps, but has difficulties when it comes to clear explanations. In my literature study I came across a few definitions, which might help with that, of which three can be found below:

> [WS04] Scalability is a measure of an application systems ability to, without modification, cost-effectively provide increased throughput, reduced response time and/or support more users when hardware resources are added.

> [WG06] Scalability is an ability of a system to handle increased workload (without adding resources).

In light of this definition we would talk about a scalability failure if one of the following occured:

- Address space was exceeded,
- Memory was overloaded,
- Available network bandwith was exceeded,
- etc.

> [WG06] Scalability is an ability of a system to handle increased workload by repeatedly applying a cost-effective strategy for extending a systems capacity.

According to this definition, one could determine system scalability failure if a given resource got overloaded or exhausted an adding capacity to this resource would not result in a proportional ability to handle additional demand, e.g.:

- an additional processor will not contribute to meeting the higher demand if handing of that processor entails an overhead).
- a newly added server instance might not contribute to handling a higher user demand if slows down the routing process.

2.1.1 Scalability Trade-offs

Scalability is generally desired in the software systems, yet, as all architectural software decisions come at a cost [Hei10]; one must be aware of the trade-offs usually associated with it. Weinstock and Goodenough [WG06] point these out:

- performance and scalability (non-scalable system will often demonstrate degrading performance with increasing demand, but scalable systems require performance sacrifice on lower usage levels),

- cost and scalability (designing a system to be scalable up-front entails additional costs),
- operability and scalability (it is difficult for humans to operate large systems),
- usability and scalability (it may be possible to increase servable demand with limiting system's service scope - e.g. removing personalization and displaying generic, cacheable data),
- data consistency and scalability (higher scalability can be achieved if system allows for data inconsistencies).

2.1.2 Need for Scalability

As the globalization and internetization progresses, more and more systems are expected to be capable of serving millions of globally distributed users. A single server instance often cannot live up to that task and thus application needs to be divided and distributed in multiple smaller chunks. This division happens on different application layers, and different parts of the system have to communicate and synchronize with each other.

In case of many software systems (the system under test 4 included), user traffic, and with it the need for system services and resources, varies significantly. A need to be able to scale up and down dynamically in response to traffic arises from this.

Huge parts of the internet are shifting towards real-time. This trend is giving rise to new technologies for exchanging messages between clients and servers in the client-server architecture. Traditionally, client would send a request to a server and receive a response. This is hugely inefficient when there is a need for continuous bidirectional exchange of messages. As an improvement, new mechanisms for server-client communication have been introduced recently 2.3.2 to enhance the process and reduce overhead. One of them - websockets - are a huge next step on this path, but introduce new challenges. One of them is scalability of applications which make use of this technology.

2.1.3 Existing Approaches

There exist multiple scalability vectors for web applications. Different decompositions of application stack can be applied to achieve the goal of scalability. A few traditional approaches of tackling an issue like that exist already [Lei08]:

Scaling up. Increasing volume of system allocated resources. As internet is unreliable and so called "middle-mile bottlenecks" exist, a web application end-user latency and throughput experience is not fully deterministic. Traffic levels fluctuate tremendously, so the need to provision for peak traffic levels means that expensive infrastructure will sit underutilized most of the time. It is easy to implement, but costly, even extremely when you start pushing at current hardware limits.[Qve10]

Scaling out. Scaling out horizontally - increasing the number of units of resources comprising the system. Cost of hardware can be reduced dramatically this way. In a web application, one can deploy multiple instances of servers. Different types of balancing can be applied to distribuet traffic among them: application layer balancing, business load balancing, and anticipating load. The overhead of parsing requests in the application layer is high thus limiting scalability compared to load balancing in the transport layer. Client state needs to be stored in a layer shared between the webservers.[Qve10]

Content Delivery Networks. These only handle static assets. Communication in our system 4 happens mainly over websockets, which are not supported by CDNs. CDNs can be divided into Big Data Center CDNs and Highly Distributed CDNs, which put application data within end-user ISPs.[Lei08]

Peer to peer networks. An architecture different from client - server, where users communicate with each other directly. It handles adding and removing nodes to and from the network dynamically very well.

2.1.4 Scaling up vs Scaling out

Tsai et al. [THBG12] provide us with a distinction between these two notions:

Scaling up (aka. vertical scaling) means deploying an application to a stronger machine, or with a better configuration, including more computing resource, more memory, higher disk bandwidth and larger disk space.

Scaling out (aka. horizontal scaling) means deploying an application to multiple machines with similar configurations.

2.1.5 Cloud Scalability

Cloud computing has become virtually ubiquitous. All biggest internet services are either deployed to a cloud, or run their proprietary cloud systems. A lot of companies running proprietary clouds also make them available for hosting to the external clients. Netflix[1] and Spotify[2] (both deployed to Amazon's AWS) are examples of the first approach, with Microsoft(Azure)[3], Google(Google Cloud Platform)[4] and Amazon(Amazon Web Services)[5] being the example of the second. These services are typically billed on a resource utility basis.

General cloud capabilities and the cloud stack I am working with for the scope of this project is discribed in a dedicated chapter 5.6.

One of the biggest advantages of deploying one's architecture to the cloud that aforementioned companies offer is the ability to dynamically scale up and in response to application traffic changes.

As Grozev and Buyya [GB14] put it, to fully facilitate cloud capabilities, software engineers need to design for the cloud, not only to deploy to it.

2.1.6 Data Layer Scalability

Data layer scalability is an important part of system scalability. In a distributed system, multiple system agents share the data. There exist multiple strategies for operating on a shared, distributed data layer.

Data layer is often a performance bottleneck because of requirements for transactional access and atomicity - it is hard to scale out when system uses a relational data store [GB14].

CAP Theorem put forward by Eric Brewer [GL02] states that it is impossible for a distributed system to provide all of the following guarantees:

- consistency,
- availability,
- partition tolerance.

Therefore, a trade-off needs to be made, depending on stakeholder priorities, which of the three to give up.

ACID stands for Atomicity, Consistency, Isolation, Durability and is a set of properties guaranteeing reliable processing of database transactions. It has been a guideline in designing multiple database systems. The term was originaly coined by Haerder and Reuter in 1983 [HR83].

BASE stands for Basically Available, Soft State, Eventually Consistent [LD12]. Its a complement of ACID. Author claims we lack precise metrics to measure BASE aspects and thats why every system implements eventual consistency differently.

Most importantly for this research, the author [LD12] explains *MySQL Cluster* - it performs much like an ACID database but with the performance benefits of a cluster. Aside from that,

[1] https://aws.amazon.com/solutions/case-studies/netflix/
[2] https://aws.amazon.com/solutions/case-studies/spotify/
[3] https://azure.microsoft.com/
[4] https://cloud.google.com/
[5] https://aws.amazon.com/

MySQL Replication can be put to use. It can be configured at multiple topologies, not only the basic master-slave; consistency differs per configuration.

Many solutions and strategies for dealing with database scalability have emerged, including *NoSQL* and *NewSQL* databases, data replication and sharding [Amz02].

Cooper et al touch on that subject in their work [CSE+10]. They claim that in scaling out the database layer one should aim for elasticity (dynamically adding capacity to a running system) and high availability. These are hard to achieve using traditional database systems. They show that new protocols are being developed to address that issue, such as that: *two-phase commit protocol* (provides atomicity for distributed transactions) and *paxos*.

The authors [CSE+10] also give an overview of different database systems, including *PNUTS*, *BigTable*, *HBase*, *Cassandra*, *Sharded MySQL*, *Azure*, *CouchDB*, and *SimpleDB*.

In their work [CSE+10], we can find enumeration of classic data-related scalability trade-offs:

- read performance and write performance,
- latency and durability,
- synchronous and asynchronous replication,
- data partitioning (column and row-based storage).

The technology stack I have been working with in the scope of this project consists of *MySQL* as persistent storage and *Redis* as a key-value cache. It is described in details in a dedicated chapter 4.4.

MySQL Scalability

Mysql White paper [Ora15] gives us a good overview of how scalability works in case of MySQL Cluster. Authors suggest starting the design of the scaling process by identifying which characteristic the application posseses: lots of write operations, real-time user experience, 24x7 user experience or agility and ease-of-use. The System Under Test 4 falls into the first and second categories.

They claim to support (among others) auto-sharding for write-scalability, active / active geographic replication and online scaling and schema upgrades. Geographic replication offers distribution of clusters across remote data centers, which they claim helps reduce latency (which is extremely important case of The System Under Test 4).

Authors show how MySQL Cluster is optimized for real-timeness:

- data structures are optimized for in-memory access,
- persistence of updates runs in the background,
- all indexed columns are stored in memory.

According to the white paper [Ora15], MySQL Cluster is very well-suited for on-line, on-demand scalin.

MySQL can also be used to scale in unconventional ways. Ruflin et al, in Social-Data Storage-Systems [RBR11], mention that Twitter uses MySQL as a key value store.

They show how MySQL can be scaled horizontally by both sharding and data replication. They also indicate that the more structured the RDBMS data, the harder it is to scale horizontally. MySQL is optimized for writes, since only one record in one table is touched, whereas reads can prove expensive if they contain joints, especially spreading across multiple cluster nodes. Facebook and Twitter solved it by putting a cache on top of MySQL [RBR11].

Redis Scalability

Single Redis installation is said to be of limited scalability, because of the fact that for good performance the whole data set should fit into memory [RBR11]. Redis offers cluster[6] and replication[7] solutions.

[6] http://redis.io/topics/cluster-spec
[7] http://redis.io/topics/replication

In a Redis Cluster, data is automatically sharded (not replicated) among multiple nodes. One has control over sharding; it is possible to ensure that certain data points end up on the same (or on a given) node. Redis Cluster does not guarantee strong consistency - under certain conditions the Cluster can lose writes that were acknowledged to the client. Support for synchronous writes exists, through the command *WAIT*, which highly (but not entirely) decreases the likelihood of lost writes. Nevertheless, using it is discouraged unless absolutely necessary.

Redis can also be scaled for reads using a simple replication in a single master - multiple slaves topology.

2.1.7 Websocket Scalability

An introduction into push-base communication over the internet can be found in Agarwals work - Toward a Push-Scalable Global Internet [Aga11]. The key message in the article is that push message delivery on the World Wide Web is not scalable for servers, intermediate network elements, and battery-operated mobile device clients. And yet, most of modern day websites have highly dynamic content updated even up to multiple times a minute (very much so for The System Under Test 4).

Most of internet communications happens over HTTP Running over TCP. Real-time message delivery requires an always-on connection from the server to the client. HTTP proxies have limited memory and TCP ports, are shared among multiple users. Servers need to be provisioned in order to maintain active TCP connections from large populations of user clients. These all provide challenges that need to be dealt with in scalable applications [Aga11].

A number of TCP ports available (and file descriptors available to a server process) on a server instance is an inherent scalability limitation of an application using the WebSocket protocol. Hardware limitations are different in an HTTP-based communication.

On a more positive note, Cassetti and Luz [CL12] claim that overhead introduced by the websocket protocol and websocket API is rather small as compared to other communication methods. Furthermore, one data intensive applications can achieve superior bandwidth and performance when using websockets.

My own research has proven preparing websocket-based communication scalability to be difficult and tedious. Most of the cloud scalability stack that is available and that I have been working with was designed and built to support http-based – and not websocket-based – communication. On top of that, one needs to tweak the host operating system kernel in multiple ways to increase the number of concurrent websocket connections the system can maintain. Details of the tuning process are described later 5.4.

The Websocket Protocol is described in details in a dedicated section 2.3.

2.1.8 Measuring Scalability

Measuring scalabliity proves a challenge, since there is no single physical quantity or unit which the community has accepted as a scalability measure.

Measuring and researching in a cloud environment is difficult, because, by definition, the researcher has no control over the hardware that his system is running on. Clouds are also inherently non-deterministic. Nevertheless, as Sobel et al. describe in their work [SSS+08], even in cloud computing environments, where researchers have little control over network topology or hardware platform, understanding the performance bottlenecks and scalability limitations imposed by the offered infrastructure is valuable.

As this is a crucial topic to my research, this topic is explored in detail in a dedicated chapter 6.

2.2 Geographical Distribution

With the rise of global internet services, single applications have to server users who are distributed around the whole globe. With the distribution of the users, comes distribution of the data. Multiple solutions exist that, through data distribution, aim at improving the *Quality of Service*.

The topic is nicely introduced by Tom Leighton in his article *Improving performance of the internet* [Lei08]. He provides a few arguments why it is important to keep data as close to end users as possible.

Apart from obvious latency benefits, by doing this, one reduces the chances of suffering from a big *middle mile* provider outage. He introduces a scale to reason about internet locality.

Scale Name	Range	Latency Range	Typical Packet Loss
local	less than 100 miles	1ms	0.6%
regional	500 - 1000 miles	15ms	0.7%
cross-continent	3000 miles	50ms	1.0%
multi-continent	6000 miles	100ms	1.4%

Table 2.1: Internet Locality

As the table 2.1 shows, the longer data must travel through the middle mile, the more it is subject to congestion, packet loss, and poor performance [Lei08]. This is why companies try to locate servers close to end users (on a scale that cannot be reproduced in this research - requires finer control over infrastructure). The lowest granular scale I am able to work with within this research is *regional*. Leighton writes that big websites have at least two geographically dispersed mirror locations to improve performance, reliability and scalability.

In his work, he includes a set of guidelines to consider when performing geographical scalability:

- reduce transport layer overhead (I am achieving this with usage of the websocket protocol),
- prefetch embedded content,
- assemble pages at the edge (even on an end client machine),
- offload computations to the edge,
- ensure significant redundancy in all systems to facilitate failover,
- use software logic to provide message reliability.

Ed Howorka in his interesting paper *Colocation beats the speed of light* [How15] about trading system geographical distributions focuses on the best placement of servers for traders trading on multiple exchanges. His paper demonstrates that traders gain nothing by positioning their computer at the midpoint between two financial exchanges. He claims that every algorithm on a central machine talking to surrounding servers (users in case of the System Under Test 4) can be replaced by colocated servers (located in geographical proximity to users). Furthermore, he implies and sets out to prove that server colocation provides a better solution for high-speed applications (as opposed to using a big, centralized server located in the middle).

2.3 WebSocket Protocol

WebSockets is an independent, TCP-based communication protocol. The protocol was standardized by the Internet Engineering Task Force (IEFT) in 2011[8] and has been gaining popularity ever since. All major web browsers and mobile operating system support it network[9].

2.3.1 Technical details

Every WebSocket communication has to start with an opening handshake. WebSocket us a protocol related to HTTP, in a sense that its handshake is interpreted by HTTP servers as an *Upgrade request*. Websocket also operates on the same ports as HTTP (80 and 443), so as not to get the communication blocked by all the internet's intermediaries configured e.g. to allow exclusively HTTP traffic.

A sample handshake included in the client's request is depicted below. It is a simple `GET` request. A set of random bytes - `Sec-WebSocket-Key` - needs to be included.

[8] https://tools.ietf.org/html/rfc6455
[9] http://caniuse.com/#feat=websockets

```
GET HTTP/1.1
Upgrade: websocket
Connection: Upgrade
Host: echo.websocket.org
Origin: http://www.websocket.org
Sec-WebSocket-Key: i9ri‘AfOgSsKwUlmLjIkGA==
Sec-WebSocket-Version: 13
Sec-WebSocket-Protocol: chat
```

Listing 2.1: Websocket Upgrade Client Request

The server taes the `Sec-WebSocket-Key`, appends a globally unique identifier (GUID) string, computes a `SHA1` hash from it and `Base64`-encodes it. The computed value is sent in the response header confirming the upgrade as `Sec-WebSocket-Accept`:

```
HTTP/1.1 101 Web Socket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Accept: Qz9Mp4/YtIjPccdpbvFEm17G8bs=
Sec-WebSocket-Protocol: chat
Access-Control-Allow-Origin: http://www.websocket.org
```

Listing 2.2: Websocket Upgrade Server Response

That leads to opening a websocket (*ws://*) communication channel between the client and the server. Secure websocket connections (*wss://*) are also possible.

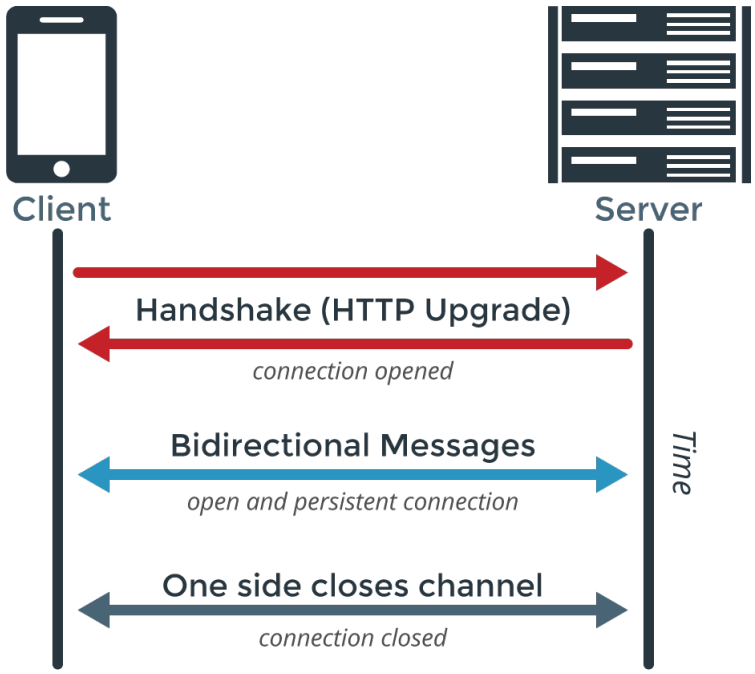

Figure 2.1: WebSocket Protocol

Typically mentioned benefits of using WebSocket protocol include reduced overhead in comparison with HTTP communication. After the initial handshake, only application-specific data travels between client server, as opposed to HTTP where every request/response contains headers, cookies, content type, content length, user-agent, server id, date, last-modified etc.

WebSocket protocol has an another advantage over HTTP on a TCP protocol level (on which both

are based) - TCP connection only needs to be opened once per WebSocket communication, whereas for HTTP it is opened for every request (which introduces overhead).

What is more, HTTP servers are often configured to persist in a log form the start and completion of every HTTP request. This entails additional CPU cycles and costly I/O operations. Unless an application is configured to log its proprietary data, this additional logging cost is much smaller in case of WebSockets (since there is only one initial request).

Configuring a server for high-scalability WebSocket communication is no easy task, as I have learned the hard way during this research. The whole process is decribed in details in a dedicated section 5.4.

2.3.2 Usage and Origins

Multiplayer online games and real-time applications with a lot of user generated content are types of application that benefit most from popularization of the WebSocket protocol.

One of the biggest advantages of websockets is that the server can easily send unsolicited messages to the client, rather than simply respond to client's requests (as is the case with HTTP protocol). Before websockets, other solutions were being developed to enable bi-directional client-server communication, yet none of them has gained such popularity and is considered as elegant.

Before websockets, the applications mentioned above had to resort to other, less efficient ways to facilitate bi-directional client-server communication.

Agarwal presents us with an overview of protocols previously used for the same kind of communication [Aga11]:

AJAX - asynchronous JavaScript and XML, a request/response model; when originally introduced, it was a huge improvement, since the page didnt have to be refreshed anymore to get new data. Still used widely, but for a slightly different purpose than the WebSocket Protocol.

Long Polling - consists of the client sending a single request and the server waits until it can provide the response (or times out), which does not create much traffic but uses a lot of server resources. Connections are artificially kept open and clients need to reconnect periodically.

Short Polling - also called AJAX-based timer, consists of the client repeatedly (timer is used to regulate that) sending the same request to the server (polling), which creates a lot of useless traffic.

webRTC - Web Real-Time Communication, enables audio, video and text communication between users using web browsers [10].

Server-Sent Events - a stream of events generated by the server, to which a client subscribes. Communication is impossible in the other direction[11].

[10] http://www.webrtc.org/
[11] https://developer.mozilla.org/en-US/docs/Server-sent_events/Using_server-sent_events

Chapter 3

Benchmarking

"If you're not keeping score, you're just practicing."
— Vince Lombardi, American Footbal Player

As this projects concerns benchmarking different architecture decompositions of The System Under Test 4, it is important to define and describe the benchmarking process.

Benchmarking - the term is most commonly defined as an execution of a set of operations against a given object / program, in order to determin it's relative performance. Usually, it consists of a set of standardized tests. The same term is often used to describe benchmarking programs themselves. It is crucial for researchers, tool developers and users [BLW15].

As James Bornholt describes in his article [Bor14], computer science research is among the most non-deterministic. A huge risk of omitted-variable bias exists. This is due to the fact that computer systems have a tremendous number of dynamic parts (both in hardware and software). It is virtually impossible for researches and other experts to have complete knowledge of every bit of the system they are working with. Controlling all the involved variables is insurmountable.

Furthermore, many of the system parts factors are non-deterministic; e.g. networks and multi-threading. Because of that, researchers need to be very careful when attempting to draw statistically meaningful conclusions from their research.

Benchmarking a distributed system is even more complex [Bor14]. If it is hosted in a cloud environment, the control over the hardware is in the hands of the cloud provider, not the researcher. This is situation I am in performing this research.

Bornholt [Bor14] lists some of the many environmental factors that can have influence on the performance of a computer system. The most striking example is the *linking order*, which can significantly bias the results of a benchmark. As UNIX systems are highly customizable, it is important to keep the values of environmental variables as consistent as possible across benchmarking sessions.

Author also states a need for the scientific community to unify and stardize the tools for benchmarking systems. A lot of researchers rebuild their infrastructures and build their scripts anew on every project. This is where I hope to contribute to the scientific and open source communities with releasing all the possible parts of the tooling build for the purpose of this project (decribed in details later 5.10).

For a benchmark to be depandable, results need to be reproducible [Bor14]. That means the same results can be obtained reruning the benchmark later on a machine with the same hardware and the same software versions. Reproducibility of experimental results requires measurements to be reliable. A measurement can be called reliable, if the method guarantees accuracy (small systematic and random measurement error, i.e., no bias or volatile effects, resp.) and sufficient precision.

Beyer et al [BLW15] list well-established benchmarks for specific tasks:

SPECi - The Standard Performance Evaluation Corporation[1]. They provide benchmarks for CPUs, Graphic Cards, Java Client/Servers, Mail Servers and others.

[1] https://www.spec.org/

TPC - Transaction Processing Performance Council[2]. They provide benchmarks for Transaction Processing (OLTP), Decision Support, Virtualization and Big Data.

nlrpBENCH - Natural Language Requirements Processing[3]. They provide benchmarks for Requirements Engineering.

In case of the system being benchmarked in this research 4, it is hard to reuse any of these since the performance depends on a custom set of operations spreading across multiple system layers.

Authors of "Benchmarking cloud serving systems with YCSB" [CSE+10] point out that it is important to have random distributions for benchmarking loads - *uniform*, *zipfian*, *latest*, *multinomial*, etc.

From my own experience, I have drawn that against a complex system like our utilizing a wide stack of technologies, benchamrking tool should do a few "dry runs" in order to give caches a chance to fill up, internal tables to get populated or cloud comopotents to be warmed up (e.g. Elastic Load Balancers in case of our stack 5.6).

One also has to be aware of what the potential congestions are. For instance, without properly tuning the operating system's TCP stack, one has to wait a few minutes between load testing runs to let all the TCP sockets return to a usable state (that is exit the TIME_WAIT state, unless the operating system is tuned to reuse such sockets). Details of the process of tuning are desciber in detail later 5.4. In the process of testing, I have also realized our API Server takes a substantial amount of time to handle disconnectis by huge number of clients (test suite disconnects all of them at the same time). That means there needs to be a non-zero pause between load test suite reruns.

3.1 Benchmarking Crimes

Gernot Heiser prepared a list [Hei10] of *benchmarking crimes* which should be avoided when attempting to benchmark a system.

Selective benchmarking

1. Not covering the full evaluation space.
2. Not evaluating potential performance degradation.
   (a) `Progressive Criterion` - performance actually does improve significantly in area of interest.
   (b) `Conservative Criterion` - performance does not significantly degrade elsewhere.

   For a benchmark to be reliable, both criterions need to be demonstrated. I trust the models that have been selected to reason about system properties 6.3 are living up that task.

   As an economist Mielton Friedman uded to put it, *there's no such thing as free lunch*[4].

   In this context, we can iterpret it in the following way. Techniques improving performance in some capacity usually entail an additional cost (extra bookkeeping, caching). As Heiser writes: "This is really at the heart of systems: it's all about picking the right trade-offs" [Hei10]. In our case the potential improved performance comes at a cost of increased data transfers between data centers (and therefore costs) and simply increased complexity of certain system layers.
3. Subsetting a benchmark without strong justification. A warning sign that one might sin in this way:
   - "we picked a representative subset", "typical results are shown" - reads as "we cherry picked the data to fit our expected results".

[2] http://www.tpc.org/
[3] http://nlrp.ipd.kit.edu/
[4] http://www.amazon.com/Theres-Such-Thing-Free-Lunch/dp/087548297X

If benchmarking is performed on a subset of a system, a strong justification needs to be as to why a given subset of funcionalities / components was selected. I am benchmarking a real-life system use case.

4. Selective data set hiding deficiencies. Luckily, more and more statistical tools are being developed to detect that [5].

Using the same dataset for calibration and validation

The way to avoid this is to calibrate the system first (using calibration workload). Then, one should use evaluation workload to show how accurate the model is. Both workloads need to be disjoint.

Providing o indication of significance of data

An example of this fallacy would be providing raw averages, without any indication of variance. At least standard deviations must be quoted. If in doubt, Heiser recommends using student's t-test to check significance [Hei10].

Benchmarking of simplified simulated system

I am performing the tests on the full system deployed in the exact same environment as the production system.

Using inappropriate and misleading benchmarks

The tests are being performed on different versions of the proprietary system. The business goal of the host organization of the research is to find the best architecture - the benchmarking proccess is supposed to help with this. There is no incentive to be dishonest.

Unfair benchmarking of competitors

The aforementioned applies.

Providing relative numbers only

Heiser suggests this reads as "I am covering that the results are really bad or irrelevant" [Hei10]. In the case of this research, some results need to stary relative since this is what the selected scalability measurement models 6.3 require. Absolutes might not make sense in this context.

Providing no proper baseline

Often the state-of-the-art solution can be used as a baseline. I believe the solution we picked as a baseline (the simplest architecture decomposition) is a good one. The research leading up to the test did not yield any data to prove that statement wrong.

Using arithmetic mean for averaging across benchmark scores

Wallace and Fleming [FW86] point out that this is an incorrect way to approaching averaging. The proper way to do this (i.e. arrive at a single figure of merit) is to use the *geometric mean of the normalised scores.*

[5] http://dataskeptic.com/epnotes/ep55_detecting-cheating-in-chess.php

Chapter 4

The System Under Test

A system under test can be best described as a *real-time, dynamic, parimutuel prediction market.* It is a platform with a publicly exposed RESTful and WebSocket APIs [1], through which system clients can connect their websites and mobile applications to access funcionality. The scope of this project concerns the WebSocket API.

The core functionality of the platform is to allow users to make predictions on (near) future events. Users can also ask their own questions regarding these events and send them out globally or to their friends. The events can literally be anything - users' imagination is the limit. A simple use case of the system would be a live football match or an e-game match broadcast on a streaming service like *twitch.tv*[2]. Questions can also be automatically generated based on an incoming stream of events regarding the e-game/sports match. The stream can consist of an "admin" inputting the events manually through the system's admin interface, or, if a game/match has an automatic stream of data available, it can also be connected to the platform.

What is important for this research, is that questions (and predictions users make on them) are organized as parimutuel pools. That means that every prediction in the pool affects the future distribution of winnings. Pennock describes it in details in his work on dynamic parimutuel markets [Pen04].

In order to keep the users informed on what the current distribution is (and thus what are the multipliers for each of the available options), the updated distribution needs to be broadcast as quickly as possible to all interested users (and reach them around the same time). In a sense, every user is therefore both a consumer and a producer of data. In case when users are globally distributed this provides a set of data-distribution and latency related challenges on which I focus in this research.

4.1 Sample Use Case

As mentioned above, one of the simplest use cases of the platform is a live football match. A client, who has connected his application to the platform can now offer users real time questions and placing predictions on them.

A case illustrating the need for the research well would be *UEFA Champions League*[3] final, for instance between *Real Madrid* and *FC Barcelona*. These events generate viewership in the range of hundreds of millions [4].

Let's assume we have 10 million users connected to the platform throughout the game. Most of them come from Spain and the rest of Europe (65%). Since the time-zone suits football-loving South Americans well, a lot of users are connected from there too (20%). The platform also hosts some North American fans (5%), with the rest comprised of hard-core Asian (9%) and Australian (1%) fans.

[1] http://instamrkt.github.io/slate/
[2] http://www.twitch.tv/
[3] http://www.uefa.com/uefachampionsleague/
[4] http://www.uefa.com/uefachampionsleague/news/newsid=2111684.html

The company providing a real-time data feed for the game have their servers located in a UK-based data center.

This raises a few interesting questions that are explored within this research:

1. Where should WebSocket server instances, which distribute messages, be spun up for minimal user-platform latencies and optimal general system performance?[5]
2. How big should the instances be?
3. What should be the geographic distribution of the instances? How should users be routed to them?
4. What should be the data-layer architecture? Should the data be centralized or distributed? Distributing using replication, sharding, or some other clustering technique?

4.2 Users of the System

The users have a possibility to connect to the platform from any mobile and desktop device that supports WebSocket communication protocol. That means they can use the platform during live games in stadiums (more and more of which provide good network connectivity), watching sports on TV or e-gaming streams from their couch or in bars and other gatherings with their friends.

That means the platform has to be capable of working efficiently not only on cable and strong wi-fi networks, but also on 4g and 3g networks to facilitate mobile usage. As the experts at AT&T inform[6], latency is more of an issue or wireless networks than on wired ones. This is because wireless network is more constrained in size and scope than a wired one. The wireless connections there need to be managed more carefully. Most usage is expected to come from mobile networks (which often drop), so reconnecting them quick to the right instance is of huge importance.

Due to the nature of the platform, the demand is extremely lumpy and comes in a few-hour-long spikes for the big events.

4.3 Basic Architecture

The most basic architecture, in its non-distributed form, of the main WebSocket API system components is depicted on the figure 4.1.

The system consists of the following units:

Event Server - receives and processes the external game/match related stream of events. Notifies the WebSocket API server so that it can broadcast the updates to the users.

Resolver - upon an arrival of a new event, collects all the outstanding pools related to it, resolves accordingly, calculates winnings, updates accounts. Notifies the WebSocket API server so that it can broadcast the updates to the users.

WebSocket API Server - handles all end-user communication. Receives requests, sends out response and broadcasts game, prediction pool and all other updates. User's subscribe to game-specific channels in order to receive updates.

4.4 Technology Stack

The platform is written mostly in Python `2.7.6`. Data storages used by the platform consist of Redis `2.8.10` as a caching solution and Mysql `2.8.6` as persitent storage. A sample client application is written in Ampersand.js[7]. Cloud stack used to which the application is deployed is fully described later 5.6.

[5] Optimal as defined in the Scalability Measurements chapter 6.

[6] http://developer.att.com/application-resource-optimizer/docs/best-practices/multiple-simultaneous-tcp-connections

[7] http://ampersandjs.com/

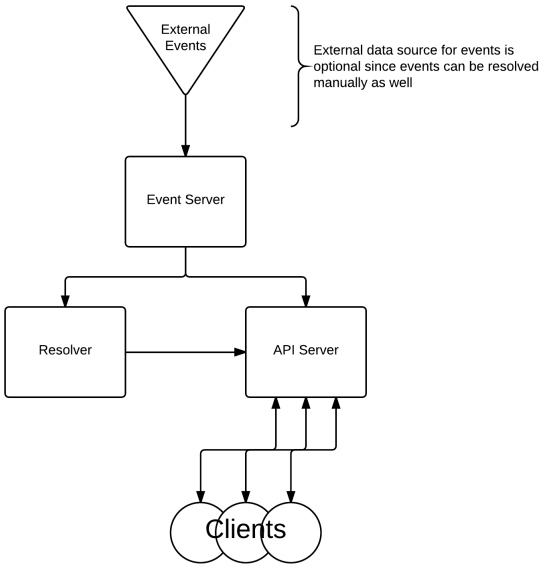

Figure 4.1: Basic System Architecture

4.5 Unique Aspects of The System

There are multiple unique aspects of the system that make this research interesting:

- Critical data is communicated using the WebSocket protocol.
- Users receiving data shared locally should receive it at the same time as data shared globally (with as small a latency as possible). This creates a need for globally consistent and manageable latency between end user and the system.
- Connected users (who serve both as consumers and producers of data) and sources of data are geographically changing. Users geographical center of mass is changing for each peak of demand.
- Demand is extremely lumpy. This creates a need for being able to quickly the system up and down.
- New data is generated constantly by the users so caching the content for geographical distribution is difficult.

Chapter 5

Experiment Outline

1. Baseline architecture on a local network.
2. Baseline architecture deployed in the cloud.
3. Improved architecture in the cloud.

All of them measured with the same set of metrics with a prepared framework for gathering them. Details below.

5.1 Goal of the experiment

The goal of the project is to design a scalability framework for a real-time persistent websocket distributed application (the system). The core researched topic will be whether a systems awareness of clients geographical distribution can improve the system performance according to selected metrics, in comparison with traditional approaches.

The goal of the project is to see if the proposed architecture decomposition can perform better (quantitatively, according to the selected metric model) than the baseline architecture in serving geographically dispersed clients. Approaches used to scale simple http applications cannot always be translated to websocket applications since the communication protocol differs. Websockets put a different kind of strain on the server machines since these need to keep the connection opened on a port for a prolonged period of time rather than simply open, server and close (as is the case with http). Along the way, an answer needs to be found what level of decoupling provides best performance on each layer of a stateless persistent system. One of the properties of a system of that kind is that key value stores come under heavy load since this is where the state resides. A good solution for distributing (sharding / replicating) these also needs to be found. Same goes for persistent storage. TODO: REFERENCE CURRENT APPROACHES HERE.

5.2 Baseline Architecture

TODO: describe here.

5.3 Load Testing Framework

all the analyzed tools, list also in lab notes from may 11 [FOR EACH WHY WASN'T SATISFACTORY] jmeter, thor (low extensibility which we needed), autobahn (max available connections but no further control), gatling-websocket, http://www.opensourcetesting.org/performance.php tsung - looked promising, highly scalable erlang, but hard to understand what was going on, community was not active enough, wsbench

5.4 Kernel tuning

5.4.1 Measurements

etsy, logster, graphite etc.

5.4.2 Linux TCP stack

http://serverfault.com/questions/48717/practical-maximum-open-file-descriptors-ulimit-n-for-a-high-volume-system the number of client connections that a server can support has nothing to do with ports in this scenario, since the server is [typically] only listening for WS/WSS connections on one single port. I think what the other commenters meant to refer to were file descriptors. You can set the maximum number of file descriptors quite high, but then you have to watch out for socket buffer sizes adding up for each open TCP/IP socket. MAKE SURE THIS IS CONSISTENT THROUGHOUT THE PAPER. If the file descriptors are tcp sockets, etc, then you risk using up a large amount of memory for the socket buffers and other kernel objects; this memory is not going to be swappable.

[http://stackoverflow.com/questions/4852702/do-html-websockets-maintain-an-open-connection-for-each-client-does-this-scale/25340220#25340220] Each TCP connection in itself consumes very little in terms server resources. Often setting up the connection can be expensive but maintaining an idle connection it is almost free. The first limitation that is usually encountered is the maximum number of file descriptors (sockets consume file descriptors) that can be open simultaneously. This often defaults to 1024 but can easily be configured higher.

all lab notes from 08.05 all websocket updates http://serverfault.com/questions/48717/practical-maximum-open-file-descriptors-ulimit-n-for-a-high-volume-system Also, and very important, you may need to check if your application has a memory/file descriptor leak. Use lsof to see all it has open to see if they are valid or not. Don't try to change your system to work around applications bugs.

Although the causes for such symptoms can vary, there's one scenario that can cause a complete lock of systems handling a very large number of web requests per second without any hint of what's going on: TCP/IP port exhaustion. [https://www.outsystems.com/forums/discussion/6956/how-to-tune-the-tcp-ip-stack-for-high-volume-of-web-requests/]

describe close_wait, time_wait etc

only 65k per (tcp 16 bit) ip address, one can add virtual ips to machines and up the number this way (recent lab notes)

increasing the number of clients The solution is more quadruplets5. This can be done in several ways (in the order of difficulty to setup): use more client ports by setting net.ipv4.ip_local_port_range to a wider range, use more server ports by asking the web server to listen to several additional ports (81, 82, 83,), use more client IP by configuring additional IP on the load balancer and use them in a round-robin fashion, use more server IP by configuring additional IP on the web server6.

But there are ways to tune the TCP/IP stack to reduce the impact of this problem, allowing the system to take advantage of all resources at its disposal. Basically, we can tune several TCP/IP stack parameters, but in this context, there are 2 that really make the different: The time that the ports is in "waiting" status since it was released from the application, and it's in fact released by the system for reuse. It's called the TIME WAIT The maximum number of ephemeral ports that the system can use, from the total pool of 65535 available ports. Let's just call it RANGE EPHEMERAL PORTS Reduce the TIME_WAIT by setting the tcp_fin_timeout kernel value on /proc/sys/net/ipv4/tcp_fin_timeout, using the command echo 30 ¿ /proc/sys/net/ipv4/tcp_fin_timeout to set it to 30 seconds.

Increase the range of ephemeral ports by setting ip_local_port_range kernel value on /proc/sys/net/ipv4/ip_local_port_r
using the command echo "32768 65535" ¿ /proc/sys/net/ipv4/ip_local_port_range, this will set the port range from 32768 to 65535.

A common misunderstanding is that a server cannot accept more than 65,536 (216) TCP sockets because TCP ports are 16-bit integer numbers. First, the number of ports is limited to 65,536, but this limitation applies only to a single IP address. Supposing that we are limited by the number of ports to have more than 65,536 clients, then adding more IP addresses to the server machine (either by adding new network cards, or simply by using IP aliasing for the existing network card) would solve the problem (even if, for opening 12 million client would need 184 network cards or IP aliases on the server

machine). In fact, the misunderstanding comes from the fact that the server does not use its listening IP address and a different ephemeral port for each new socket to distinguish among the sockets, but it uses the same listening IP address and the same listening port for all sockets and it distinguishes among sockets by using the IP address and the ephemeral port of each client. Therefore, MigratoryData Server uses a single port to accept any number of clients and optionally it uses another few ports for JMX monitoring, HTTP monitoring, etc https://mrotaru.wordpress.com/category/websockets/

max descriptors per process: Because one cannot increase the maximum number of socket descriptors per process to a value larger than the current kernel maximum (fs.nr_open) and because the kernel maximum defaults to 1048576 (10242), prior to running the ulimit command, we increased the kernel maximum accordingly as follows: echo 20000500 ¿ /proc/sys/fs/nr_open CHECK LAB NOTES RECENT TESTS, A LOT https://mrotaru.wordpress.com/category/websockets/

The kernel value parameters aren't saved with these commands, and are reset to the default values on system reboot, thus make sure to place the commands on a system startup script such as /etc/rc.local. https://www.outsystems.com/forums/discussion/6956/how-to-tune-the-tcp-ip-stack-for-high-volume-of-web-requests/

look for _what can limit concurrent sockets?_ in lab notes

The number of connections wstest can open on a server is limited by the number of ephemeral ports on the machine on the outgoing interface / IP. Something like 64k at most. If you need to test the server with more connections, currently you will need to run multiple instances of wstest (on different machines). from: http://autobahn.ws/testsuite/usage.html#mode-massconnect

Understanding tcp sockets: sockets are left in TIME_WAIT on the server when the client suite is killed which makes sense when client exits gracefully no sockets left in TIME_WAIT / CLOSE_WAIT by the server on_close() from tornado only kicks in on client close .onclose in js only kicks in when server initiates through self.close() when server terminates through self.close() sockets ends in TIME_WAIT for a minute (also makes sense according to docs) describe how you can influence that (reusing web sockets + delay time)

ulimit, nofile, fs.maxfile, net.ipv4.ip_local_port_range = 32768 65535 to /etc/sysctl.conf for ip ranges on the clients

5.5 Baseline architecture on a local network

Architecture diagrams. Technical details - local server capabilities. Load testing. LOCAL NETWORK DESCRIPTION

5.6 Cloud architecture setup

selected aws

multiple availability zones very important in a shared cloud environment - often because another customer is getting hit by a DDoS.[GB14]

In Amazons environment, Grozev and Buyya suggest using Elasticache service [GB14]. We cannot use this, also described in 5.6.

5.6.1 Route53

dns mapped to inscance ips / load balancer dns names lbr, geo, weighed rr their internal heuristics take network conditions from past weeks [VERIFY], rather amibiguous For each end users location, Route 53 will return the most specific Geo DNS record that includes that location. In other words, for a given end users location, Route 53 will first return a state record; if no state record is found, Route 53 will return a country record; if no country record is found, Route 53 will return a continent record; and finally, if no continent record is found, Route 53 will return the global record.

health checks - can be used custom-defined, but for us we can use load balancing health checks (will only route if healthy incstances behind it) [http://docs.aws.amazon.com/Route53/latest/DeveloperGuide/dns-failover-complex-configs.html]

http://docs.aws.amazon.com/Route53/latest/DeveloperGuide/Tutorials.html

5.6.2 Autoscaling

instances behind each groups. automatically scalable, connected to external redis and mysql instances. only within one region, multiple availability zones http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/Ge http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/how-as-works.html#arch-AutoScalingMultiAZ Auto Scaling attempts to distribute instances evenly between the Availability Zones that are enabled for your Auto Scaling group. Auto Scaling does this by attempting to launch new instances in the Availability Zone with the fewest instances. alarm - object that watches over a single metric (e.g. avg cpu use of ec2 instances in auto scaling group over specified time period) (OK, ALARM, INSSUFICIENT_DATA), can trigger scaling up / down policy on an autoscaling group policy variants (change ExactCapacity, ChangeInCapacity, PercentChangeInCapacity) recommendation one policy for scaling out, another policy for scaling in http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/as-scale-based-on-demand.html http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/AlarmThatSe http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/US_AlarmAtThresholdEC2.html

Alarms

avg / min / max / sum / sample count (for whole group) of cpu / disk io / network io is ¿= / ¿ / ¡ / ¡= than x for at least X consecutive periods of 5 min / 15 min / 1 hour / 6 hours

Scaling Policies

execute policy when alarm add / remove / set to X instances / and then wait Y (cooldown period)

Launch configurations

5.7 Load balancing

elb uses dns name to server a pool of lb instances in the backend (you might exceed one instance connection limit), that's why they are not provided a static ip routing: request count-based for http(s), for others (tcp which is a workaround for websockets) they use tcp [http://docs.aws.amazon.com/ElasticLoadBalancing/late elb-works.html#request-routing] The client uses DNS round robin to determine which IP address to use to send the request to the load balancer. No control over routing without load balancing groups. For example, if you have ten instances in Availability Zone us-west-2a and two instances in us-west-2b, the traffic is equally distributed between the two Availability Zones. As a result, the two instances in us-west-2b serve the same amount of traffic as the ten instances in us-west-2a. Instead, you should distribute your instances so that you have six instances in each Availability Zone. To distribute traffic evenly across all back-end instances, regardless of the Availability Zone, enable cross-zone load balancing on your load balancer. However, we still recommend that you maintain approximately equivalent numbers of instances in each Availability Zone for higher fault tolerance. [http://docs.aws.amazon.com/ElasticLoadBalancing/latest/DeveloperGuide/how-elb-works.html#request-routing] only 64k connections possible per ELB instance TODO: VERIFY THAT stickiness only works for HTTP/HTTPS protocols Once you have a testing tool in place, you will need to define the growth in the load. We recommend that you increase the load at a rate of no more than 50 percent every five minutes. B.oth step patterns and linear patterns for load generation should work well with Elastic Load Balancing. If you are going to use a random load generator, then it is important that you set the ceiling for the spikes so that they do not go above the load that Elastic Load Balancing will handle until it scales (see Pre-Warming the ELB). [https://aws.amazon.com/articles/1636185810492479#pre-warming]

5.8 Data Layer

Technologies we used described in 4.4. We used Mysql and redis as a caching solution.

5.8.1 Amazon Relational Database Service

Does not support mysql cluster as of July 2015. Support coming but not released yet. Offers mysql read replicas. With a single master. That obviously scales only reads You can create a MySQL Read Replica in a different region than the source DB instance to improve your disaster recovery capabilities, scale read operations into a region closer to end users, or make it easier to migrate from a data center in one region to a data center in another region.

5.8.2 Elasticache

doesn't support crossregion (you cannot connect to an elasticache cluster from outisde of the region)

The above explain why, for a scaled out solution we had to roll with our custom setup. We were suggested using technologies like DynamoDB and Kinesis, but out of scope.

5.8.3 Baseline architecture deployed in the cloud

5.8.4 Improved architecture deployed in the cloud

5.9 System Scaling Delay

Delays: up to 2 minutes to get metrics, up to ????? minutes for instance to be accessible to clients + ???? for DNS changes to propagate (TTL set to 60 seconds but TODO VERITYF) Unfortunately there are a large number of (misbehaving) DNS servers out there that dont properly obey TTLs on records and will still serve up stale records for an indefinite amount of time. [http://engineering.chartbeat.com/2014/01/02/part-1-lessons-learned-tuning-tcp-and-nginx-in-ec2/]

5.10 Experiment deliverables

open sourced code for: using scalability models managing the cloud (built on boto) remote execution of load test and metric collection

Chapter 6

Scalability Measurements

Here we describe the metrics we settled for.

Most of what is necessary for the purpose of this research has been covered by Pushkala Pattabiraman et al [MMPROJ3]. They realized that cloud computing and its measurement provides a new set of challenges when it comes to measuring performance testing, as opposed to measuring performance of traditional software systems. They list some key points for measuring cloud applications, among them we can find: validating and ensuring the elasticity of scalability and evaluating utility service billings and pricing models. The latter is also important in my case since cost is one of the driving factors in assessing the scalability of my system. A question they raise regarding this is: How to use a transparent approach to monitor and evaluate the correctness of the utility bill based on a posted price model during system performance evaluation and scalability measurement?. The authors divide the performance indicators into three groups: computing resource (CPU, disk, memory, networks) - they can be helpful in establishing baseline architecture in my case, workload indicators (connected users, throughput and latency), performance indicators - processing speed, system reliability and scalability based on the given QoS standards. For each they propose formal models with pluggable values and graphic representations (BELOW). On top of that, the research contains a case study performed in the Amazon EC2 environment [MMPROJ3]. Cloud limitations need to be taken into account. One needs to be aware of hidden costs (e.g. autoscaling service is free on EC2, but it requires cloudwatch, which is not). The authors also advise to pay attention to inconsistencies in performance and scalability data [MMPROJ3].

many others: There is much more work related to the general scalability of distributed systems. Srinivas and Janakiram in their work [MMPROJ5] mention a metric evaluating scalability as a product of throughput and response time (or any value function) divided by the cost factor. They propose another model considering scalability as a function of synchronization, consistency, availability, workload and faultload. It aims on identifying bottlenecks and hence improving the scalability. The authors also emphasize the fact of interconnectedness of synchronization, consistency and availability. Jogalekar and Woodside [6] propose a strategy-based scalability metric based on cost effectiveness (a function of system's throughput and its quality of service). It separates evaluation of throughput or quantity of work from QoS (which, according to the authors, can be any suitable expression).[MMPROJ6] PASA[MMPROJ17]

[SJ05] Srinivas and Janakiram in their work mention a metric evaluating scalability as a product of throughput and response time (or any value function) divided by the cost factor - include formula [JW00] work on Performance, Reliability, Scalability testing [last most relevant], In their research, we also find some interesting, relevant to my case, advice concerning reliability testing. Authors offer suggestions on how to identify potential problem areas using these sample criteria: performance bottlenecks that can be attributed to size of data system performs with stability beyond x hours from startup time (as defined by the following) under the given loading conditions and does not change more than y percent with regard to: system throughput measured by the total number of requests served per second cycle time for different workflow number of threads of different processes in the configuration virtual memory utilization of different processes P-scalabilityP(k1,k2) - take from [mmproj]

6.1 Expected Answers

6.2 Selected metrics

latency (messaging + handhsaking) (our suite) sustainable concurrent websocket connectoins (our suite) infrastructure cost (per unit of time, supporting the same number of users) (manual calculation) dropped connections (our suite) cpu usage (cloudwatch) memory usage (our custom server metrics, pidstat) network in network out (cloudwatch) iops reads + writes (bots number and byte values, cloudwatch)

to calculate manually: percentage of available ports used? how long to run on a x euros

availability (SLAs???????) architecture reaction speed to changes in demand (scaling up and scaling down) (????)

CRAM METRICS TODO: review

6.2.1 EC2 Available metrics

collectible every minute (in detailed mode), by default every 5 minute, paid additionaly - delay up to 2 minutes the current EC2 cloud technology does not provide any CloudWatch API to monitor the allocation and utilization of memory for EC2 instances (we do it on our own) limits exist (10 metric, 10 alarms, 1,000,000 million requests, 1000 SNS email notifications per month for free no limits on custom metrics up to 5gb of incoming data logs for free up to 5gb of data archiving for free) but we don't expect to hit them. small delay (up to 2 minutes, that's what we experienced - delays in autoscaling)

alarms can be configured based on this - http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-cloudwatch.html data not aggregated across regions [lab2]

Instance Specific

[Lab2 - http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/cloudwatch_concepts.html] custom possible available statistics for each: min, max, sum, avg, sampleCount (count of data points but research that) cpu utilization network in / out (in bytes) disk read / disk write (number of ops + bytes) status checks

Cloud (group) Specific

Collected from Elastic Load Balancer for all instances connected.

Custom Metric Collection

pidstat latencies by our custom client suite for network we looked ad ntop, nethogs, nload, vnstat, wireshark but amazon provides this data.

ss -s netstat -s

plugged into statsd and graphite for real time monitoring

6.3 Selected models

[MMPROJ3] With CRUMs, SPMs etc. four different types of needs: resource utilization and allocation measurement performance measurement under allocated computing resources scalability measurement in different cloud infrastructures cost driven evaluation based on a pre-defined price model

additional values we want to capture to proposed in the model: cost time to scale up / down What we will use to compare architectures: SCM (allocated resources) or SEC (used) (1/cost) as an additional metric to SEC? or amount of s you can run on a 100bucks

Results of all setups get plugged into SCM / SEC and then ESS.

6.4 Baseline, Local network

Lab notes from 11 of May go here:
Basic metrics

- cpu, memory, disk usage (pidstat / CloudWatch)
- network i/o (wireshark, list others analyzed)
- latency, throughput, concurrent connections, messages dropped?

6.4.1 Load Testing

http://stackoverflow.com/questions/3871886/ssl-and-load-balancing
elb uses dns name to server a pool of lb instances in the backend (you might exceed one instance connection limit) in order for the full range of ELB machine IP addresses to be utilized, make sure each [client] refreshes their DNS resolution results every few minutes. One test client equals one public IP address. ELB machines seem to route all traffic from a single IP address to the same back-end instance, so if you run more than one test client process behind a single public IP address, ELB regards these as a single client. Use 12 test clients for every availability zone you have enabled on the ELB. These test clients do not need to be in different availability zones they do not even need to be in EC2 (although it is quite attractive to use EC2 instances for test clients). If you have configured your ELB to balance among two availability zones then you should use 24 test clients. Each test client should gradually ramp up its load over the course of a few hours. Each client can begin at (for example) one connection per second, and increase its rate of connections-per-second every X minutes until the load reaches your desired target after a few hours. - WE CANNOT DO THAT
Tools described here [**?**]

Users Distribution

The authors suggest choosing randomly when generating load as to which operation to perform, on what data size etc. They suggest using different random distributions: uniform, zipfian, latest, multinomial[CSE+10]. (as mentioned in bemncharking 2.2 section)
Another distribution is suggested by Grozev and Buyya [GB14] - Poisson distribution with a constant mean.

6.5 Baseline, Deployed in the cloud

6.6 Improved, Deployed in the cloud

Chapter 7

Experiment results

Chapter 8

Evaluation

8.1 Statistical Significance

http://technology.stitchfix.com/blog/2015/05/26/significant-sample/

8.2 Threats to validity

ec2 whacky, control over hardware released to amazon (can be dedicated machines) shared environment, someone else might get ddosed or sth - you have no control over that network is inherently non-deterministic, load tests needed to actually test availability

Chapter 9

Conclusions

Chapter 10

Further Work

Chapter 11

Everything below that needs to be removed (except for bib), Front Matter

The first thing is to connect the class by saying:

```
\documentclass{uvamscse}
```

11.1 Title

Specify the title of the thesis with \title and \subtitle commands:

```
\title{MetaThesis}
\subtitle{A Thesis Template Leading by Example}
```

Any thesis can survive without a \subtitle, but the \title is mandatory.

11.2 Author

Introduce yourself with \author and \authemail:

```
\author{Vadim Zaytsev}
\authemail{vadim@grammarware.net}
```

Again, \authemail is not mandatory. If you need anything fancier, just put it inside \author.

```
\author{Vadim Zaytsev\footnote{Yes, that one.}}
```

The footnote would be printed on the bottom of the title page, and will be referred to by a symbol, not by a number as any footnotes within the main document body.

11.3 Date

By default, the date inserted in your PDF is the day of the build, e.g., "March 25, 2014". If you want it to be formatted differently or be more vague or outright fake, use \date:

```
\date{Spring 2014}
```

The argument is just a string, the format is unrestricted:

```
\date{Tomorrow. Honestly.}
```

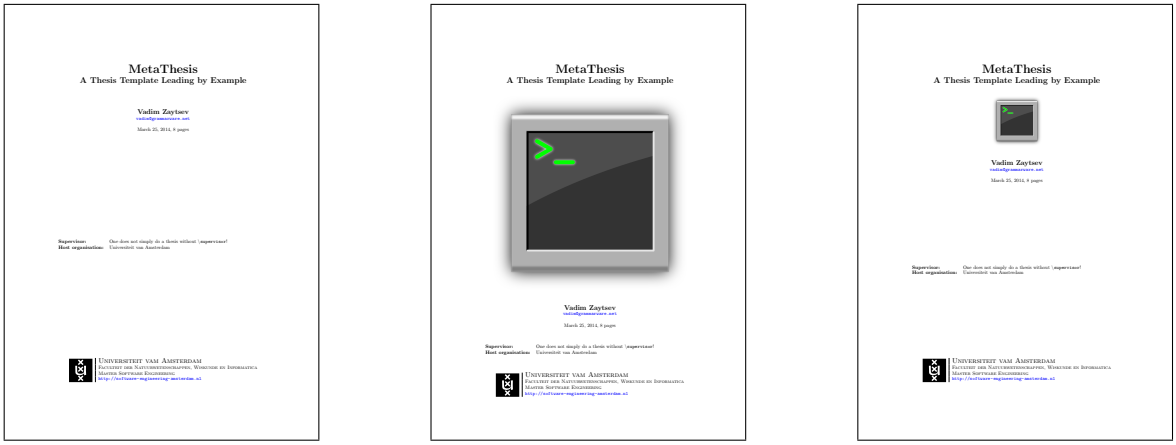

Figure 11.1: A hypothetical thesis title page without a cover picture (on the left), with an overly large one (in the centre) and with a tiny pic (on the right).

11.4 Host

If your hosting organisation is not the UvA, specify it with `\host`. The logo on the bottom of the title page will still be the UvA one, because this is the organisation guaranteeing your degree.

```
\host{Grammarware, Inc., \url{http://grammarware.github.io}}
```

NB: footnotes will not work, unless you know how to `\protect` them.

11.5 Cover picture

If the first page of your thesis looks too blunt, add a picture to it:

```
\coverpic{figures/terminal.png}
```

You can even specify the picture's width as an optional argument:

```
\coverpic[100pt]{figures/terminal.png}
```

How these three options look, you can see from Figure 11.1.

11.6 Abstract

A thesis is fine without an abstract, if you do not feel like writing it and your supervisor does not feel like enforcing it. If you do want an abstract, make it with the `\abstract` command:

```
\abstract{This is not a thesis.}
```

The abstract is just like any other section of your thesis, so you can use any LaTeX tricks there. If you think that the name "abstract" is too abstract for your abstract, you can still use `\abstract` without being too abstract:

```
\abstract[Confession]{I am a cenosillicaphobiac.}
```

Kent Beck [JBB+93] proposes to have four sentences in a good abstract:

1. The first states the problem.
2. The second states why the problem is a problem.
3. The third is the startling sentence.
4. The fourth states the implication of the startling sentence.

In practice, each of these "sentences" can be longer than an actual sentence, but it is in general a good rule of thumb to condense the summary of your thesis into these four tiny messages. Do not write too much, make it tweetable.

Chapter 12

Core Chapters

The structure of your thesis is up to you and your supervisor. Whatever you do, do not consider the guidelines below as dogmas.

12.1 Classic structure

Problem statement and motivation. You describe in detail what problem the research is addressing, and what is the motivation to address this problem. There is a concise and objective statement of the research questions, hypotheses and goals. It is made clear why these questions and goals are important and relevant to the world outside the university (assuming it exists). You can already split the main research question into subquestions in this chapter. This section also describes an analysis of the problem: where does it occur and how, how often, and what are the consequences? An important part is also to scope the research: what aspects are included and what aspects are deliberately left out, and why?

Research method. Here you describe the methods used to answer the research questions. A good structure of this section often follows the subquestions by providing a method for each. The research method needs a thorough motivation grounded in theory in order to be acceptable. As a part of the method, you can introduce a number of hypotheses — these will be tested by the research, using the methods described here. An important part of this section is validation. How will you evaluate and validate the outcomes of the research?

Background and context. This chapter contains all the information needed to put the thesis into context. It is common to use a revised version of your literature survey for this purpose. It is important to explicitly refer from your text to sources you have used, they will be listed in your bibliography. For example, you can write "A small number of programming languages account for most language use [MR13]", where the following entry would be included in your bibliography:

> [MR13] Leo A. Meyerovich and Ariel S. Rabkin. Empirical Analysis of Programming Language Adoption. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA, pages 1–18. ACM, 2013. doi:10.1145/2509136.2509515.

Have a look at § 13 to learn more about citation.

Research. This chapter reports on the execution of the research method as described in an earlier chapter. If the research has been divided into phases, they are introduced, reported on and concluded individually. If needed, this chapter could be split up to balance out the sizes of all chapters.

Results. This chapter presents and clarifies the results obtained during the research. The focus should be on the factual results, not the interpretation or discussion. Tables and graphics should be used to increase the clarity of the results where applicable.

Analysis and conclusions. This chapter contains the analysis and interpretation of the results. The research questions are answered as best as possible with the results that were obtained. The analysis also discussed parts of the questions that were left unanswered.

An important topic is the validity of the results. What methods of validation were used? Could the results be generalised to other cases? What threats to validity can be identified? There is room here to discuss the results of related scientific literature here as well. How do the results obtained here relate to other work, and what consequences are there? Did your approach work better or worse? Did you learn anything new compared to the already existing body of knowledge? Finally, what could you say in hindsight on the research approach by followed? What could have done better? What lessons have been learned? What could other researchers use from your experience? A separate section should be devoted to "future work", i.e., possible extension points of your work that you have identified. Even other researchers should be able to use those as a starting point.

12.2 Reporting on replications

Here are the guidelines to report on replicated studies [Car10]:

Information about the original study

Research question(s) that were the basis for the design

Participants, their number and any other relevant characteristics

Design as a graphical or textual description of the experimental design

Artefacts, the description of them and/or links to the artefacts used

Context variables as any important details that affected the design of the study or interpretation of the results

Summary of the results in a brief overview of the major findings

Information about the replication

Motivation for conducting the replication as a description of why the replication was conducted: to validate the results, to broaden the results by changing the participant pool or the artifacts.

Level of interaction with original experimenters. The level of interaction between the original experimenters and the replicators should be reported. This interaction could range from none (i.e. simply read the paper) to them being the same people. There is quite a lot of discussion of the level of interaction allowed for the replication to be "successful", but this level should be reported even without addressing the controversy.

Changes to the original experiment. Any changes made to the design, participants, artifacts, procedures, data collected and/or analysis techniques should be discussed along with the motivation for the change.

Comparison of results to original

Consistent results, when replication results supported results from the original study, and

Differences in results, when results from the replication did not coincide with the results from the original study. Authors should also discuss how changes made to the experimental design (see above) may have caused these differences.

Drawing conclusions across studies

NB: this section contains portions of text repeated directly from Carver [Car10] and only slightly massaged. Do not do this for your thesis, write your own thoughts down.

12.3 LaTeX details

12.3.1 Environments

A LaTeX environment is something with opening and closing tags, which look like `\begin{name}` and `\end{name}`. Some useful environments to know:

`itemize`	bullet lists
`enumerate`	numbered lists
`description`	definition lists
`center`	centered line elements
`flushright`	right aligned lines
`flushleft`	left aligned lines
`tabular`	table
`longtable`	multi-page table (needs the `longtable` package)
`sideways`	rotates some text
`quote`	block quote
`verbatim`	unformatted text
`minipage`	compound box with elements inside
`boxedminipage`	compound box with elements inside and a border around it
`table`	floating table (needs to have `tabular` nested inside)
`figure`	floating figure
`sourcecode`	floating listing
`equation`	mathematical equation
`lstlisting`	pretty-printed syntax highligted listing
`multline`	mathematical equation spanning over multiple lines
`eqnarray`	system of mathematical equations
`gather`	bundled mathematical equations
`align`	bundled and aligned mathematical equations
`array`	matrix
`CD`	commutative diagrams

12.4 Listings

```
1  define(Ps1,G1,G2)
2   ⇐
3      usedNs(G1,Uses),
4      ps2n(Ps1,N),
5      require(
6        member(N,Uses),
7        'Nonterminal ~q must not be fresh.',
8        [N]),
9      new(Ps1,N,G1,G2),
10     !.
```

Listing 12.3: Code in Prolog

```
module Syntax

imports Numbers
imports basic/Whitespace

exports
  sorts
    Program Function Expr Ops Name Newline

  context−free syntax
    Function+                          → Program
    Name Name+ "=" Expr Newline+       → Function
    Expr Ops Expr                      → Expr    {left ,prefer,cons(binary)}
    Name Expr+                         → Expr    {avoid,cons(apply)}
    "if" Expr "then" Expr "else" Expr  → Expr    {cons(ifThenElse)}
    "(" Expr ")"                       → Expr    {bracket}
    Name                               → Expr    {cons(argument)}
    Int                                → Expr    {cons(literal)}
    "−"                                → Ops     {cons(minus)}
    "+"                                → Ops     {cons(plus)}
    "=="                               → Ops     {cons(equal)}
```

Listing 12.4: Code in SDF

```
1  import types.*;
2  import org.antlr.runtime.*;
3
4  public class TestEvaluator
5      public static void main(String[] args) throws Exception {
6
7          // Parse file to program
8          ANTLRFileStream input = new ANTLRFileStream(args[0]);
9          FLLexer lexer = new FLLexer(input);
10         CommonTokenStream tokens = new CommonTokenStream(lexer);
11         FLParser parser = new FLParser(tokens);
12         Program program = parser.program();
13
14         // Parse sample expression
15         input = new ANTLRFileStream(args[1]);
16         lexer = new FLLexer(input);
17         tokens = new CommonTokenStream(lexer);
18         parser = new FLParser(tokens);
19         Expr expr = parser.expr();
20
21         // Evaluate program
22         Evaluator eval = new Evaluator(program);
23         int expected = Integer.parseInt(args [2]);
```

Listing 12.5: Code in Java

```
#!/usr/local/bin/python
# wiki: BGF
import os
import sys
import slpsns
import elementtree.ElementTree as ET

# root::nonterminal* production*
class Grammar:
  def __init__(self):
    self.roots = []
    self.prods = []
  def parse(self,fname):
    self.roots = []
    self.prods = []
    self.xml = ET.parse(fname)
    for e in self.xml.findall('root'):
      self.roots.append(e.text)
    for e in self.xml.findall(slpsns.bgf_('production')):
      prod = Production()
      prod.parse(e)
      self.prods.append(prod)
```

Listing 12.6: Code in Python

Chapter 13

Literature

BibTeX is a JSON-like format for bibliographic entries. Encode each source once as a BibTeX entry, give it a name and refer to it from any place in your thesis. The bibliography at the end of the thesis will be compiled automatically from those entries that are referenced at least once, it will also be automatically sorted and fancyfied (URLs, DOIs, etc).

DOI is a digital object identifier, it is uniquely and immutably assigned to any paper published in a well-established journal or conference proceedings and can be used to refer to it. When used in a browser, it resolves to a publisher's website where paper can be obtained. Including DOIs in citations is considered good practice and lets the readers of your thesis get to the text of the paper in one click. Books do not have DOIs, only ISBNs; some workshop proceedings and most unofficial publications do not have DOIs. If you want to get a DOI assigned to your work such as a piece of code, upload it to FigShare.

Keys in key-value pairs within each BibTeX entry are never quoted, values usually are, but can also be included within curly brackets or left as is, which works fine for numbers (e.g., years). If you want to preserve the value from any adjustments (e.g., no recapitalisation in titles), use curlies *and* quotes. Separate authors and editors by "and", which will automatically be mapped to commas or left as "and"s as necessary.

13.1 Books

[GJ08] is just as good as the Dragon Book, but newer and has an awesome extended bibliography available for free.

```
@book{GruneJacobs,
  author    = "D. Grune and C. J. H. Jacobs",
  title     = "{Parsing Techniques: A Practical Guide}",
  series    = "Monographs in Computer Science",
  edition   = 2,
  publisher = "Springer",
  url       = "http://www.cs.vu.nl/~dick/PT2Ed.html",
  year      = 2008,
}
```

13.2 Journal papers

Not all TOSEM papers are hard to read [KLV05].

```
@article{GrammarwareAgenda,
  author      = "Paul Klint and Ralf L{\"a}mmel and Chris Verhoef",
  title       = "{Toward an Engineering Discipline for Grammarware}",
  journal     = "ACM Transactions on Software Engineering Methodology (TOSEM)",
  volume      = 14,
  number      = 3,
  year        = 2005,
  pages       = "331--380",
}
```

13.3 Conference papers

There is no limit to how many grammars can be used in one paper, but the current record stands at 569 [Zay13].

```
@inproceedings{Micropatterns2013,
  author = "Vadim Zaytsev",
  title = "{Micropatterns in Grammars}",
  booktitle = "{Proceedings of the Sixth International Conference on Software Language Engineering
               (SLE 2013)}",
  year = 2013,
  editor = "Martin Erwig and Richard F. Paige and Eric Van Wyk",
  volume = "8225",
  series = "LNCS",
  pages = "117--136",
  address = "Switzerland",
  month = oct,
  publisher = "Springer International Publishing",
  doi = "10.1007/978-3-319-02654-1_7",
}
```

13.4 Theses

The seventh PhD student of Paul Klint was Jan Rekers [Rek92].

```
@phdthesis{Rekers92,
 author   = "J. Rekers",
 title    = "{Parser Generation for Interactive Environments}",
 school   = "University of Amsterdam",
 year     = 1992,
 url      = "http://homepages.cwi.nl/~paulk/dissertations/Rekers.pdf",
}
```

There is also `mastersthesis` type with exactly the same structure for referring to Master's theses.

13.5 Technical reports

The original seminal work introducing two-level grammars was never published in any book or conference, but there is a technical report explaining it [vW65]. SMC, or *Stichting Matematisch Centrum*, was the old name of CWI fifty years ago.

```
@techreport{Wijngaarden65,
        author      = "Adriaan van Wijngaarden",
        title       = "{Orthogonal Design and Description of a Formal Language}",
        month       = oct,
        year        = 1965,
        institution = "SMC",
        type        = "{MR 76}",
        url         = "http://www.fh-jena.de/~kleine/history/languages/VanWijngaarden-MR76.pdf",
}
```

13.6 Wikipedia

You do not refer to Wikipedia from academic writing, it works the other way around.

13.7 Anything else

You can refer to pretty much anything (websites, blog posts, software) through `misc` type of entry [Par08]:

```
@misc{ANTLR,
 author       = "Terence Parr",
 title        = "{ANTLR---ANother Tool for Language Recognition}",
 howpublished = "Software",
 url          = "http://antlr.org",
 year         = "2008"
}
```

Bibliography

[Aga11] S. Agarwal. Toward a push-scalable global internet. In *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, pages 786–791. IEEE, 2011. doi:10.1109/INFCOMW.2011.5928918.

[Amz02] C. Amza. Specification and implementation of dynamic Web site benchmarks. In *Workload Characterization, 2002. WWC-5. 2002 IEEE International Workshop on*, pages 3–13. IEEE, November 2002. doi:10.1109/WWC.2002.1226489.

[BLW15] Dirk Beyer, Stefan Lwe, and Philipp Wendler. Benchmarking and Resource Measurement. In *22nd SPIN Symposium on Model Checking of Software (SPIN 2015)*. Springer, 2015. URL: http://www.sosy-lab.org/~dbeyer/Publications/2015-SPIN.Benchmarking_and_Resource_Measurement.pdf.

[Bor14] James Bornholt. How Not to Measure Computer System Performance. Website, November 2014. URL: https://homes.cs.washington.edu/~bornholt/post/performance-evaluation.html.

[Car10] Jeffrey C. Carver. Towards Reporting Guidelines for Experimental Replications: A Proposal. In *Proceedings of the International Workshop on Replication in Empirical Software Engineering Research*, RESER, May 2010.

[Chu67] C. West Churchman. Wicked problems. *Management Science*, 14(4):B–141–B–146, 1967. doi:10.1287/mnsc.14.4.B141.

[CL12] Oscar Cassetti and Saturnino Luz. The websocket api as supporting technology for distributed and agent-driven data mining oscar cassetti, 2012.

[CSE+10] Brian F. Cooper, A. Siblerstein, E.Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with YCSB. In *SoCC '10 Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM, 2010. doi:10.1145/1807128.1807152.

[ESSD08] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting empirical methods for software engineering research. In Forrest Shull, Janice Singer, and DagI.K. Sjberg, editors, *Guide to Advanced Empirical Software Engineering*, pages 285–311. Springer London, 2008. doi:10.1007/978-1-84800-044-5_11.

[FW86] Philip J. Fleming and John J. Wallace. How not to lie with statistics: The correct way to summarize benchmark results. *Commun. ACM*, 29(3):218–221, March 1986. URL: http://doi.acm.org/10.1145/5666.5673, doi:10.1145/5666.5673.

[GB14] Nikolay Grozev and Rajkumar Buyya. Multi-Cloud Provisioning and Load Distribution for Three-Tier Applications. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 9(13), October 2014. URL: http://dl.acm.org/citation.cfm?id=2662112.

[GJ08] Dick Grune and C. J. H. Jacobs. *Parsing Techniques: A Practical Guide*. Monographs in Computer Science. Springer, 2 edition, 2008. URL: http://www.cs.vu.nl/~dick/PT2Ed.html.

[GL02] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2):51–59, June 2002. URL: http://dl.acm.org/citation.cfm?id=564601.

[GPBT11] J. Gao, P. Pattabhiraman, Xiaoying Bai, and W.T. Tsai. Saas performance and scalability evaluation in clouds. In *Service Oriented System Engineering (SOSE), 2011 IEEE 6th International Symposium on*, pages 61–71, Dec 2011. doi:10.1109/SOSE.2011.6139093.

[Hei10] Gernot Heiser. Systems benchmarking crimes. Toolkit website, January 2010. URL: https://www.cse.unsw.edu.au/~gernot/benchmarking-crimes.html.

[How15] Ed Howorka. Colocation beats the speed of light. Website, February 2015. URL: http://edhoworka.com/colocation-beats-the-sol/.

[HR83] Theo Harder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys (CSUR)*, 15(4):287–317, December 1983. doi:10.1145/289.291.

[JBB+93] Ralph E. Johnson, Kent Beck, Grady Booch, William R. Cook, Richard P. Gabriel, and Rebecca Wirfs-Brock. How to Get a Paper Accepted at OOPSLA. In Timlynn Babitsky and Jim Salmons, editors, *Proceedings of the Eighth Annual Conference on Object-Oriented Programming Systems, Languages and Applications*, OOPSLA, pages 429–436. ACM, 1993.

[JW00] P. Jogalekar and M. Woodside. Evaluating the scalability of distributed systems. *Parallel and Distributed Systems, IEEE Transactions on*, 11(6):589–603, June 2000. doi:10.1109/71.862209.

[KLV05] Paul Klint, Ralf Lämmel, and Chris Verhoef. Toward an Engineering Discipline for Grammarware. *ACM Transactions on Software Engineering Methodology (TOSEM)*, 14(3):331–380, 2005.

[LD12] Louis Liu and Alexander Dekhtyar. CSC 560: Advanced Topics in Databases: Modern DBMS Architectures, Eventual Consistency. University website, December 2012. URL: https://wiki.csc.calpoly.edu/csc560/raw.../wiki/.../Louis_Liu_paper.pdf.

[Lei08] Tom Leighton. Improving Performance on the Internet. *ACM Queue - Scalable Web Services*, 6(6):20–29, October 2008. URL: http://queue.acm.org/detail.cfm?id=1466449.

[MR13] Leo A. Meyerovich and Ariel S. Rabkin. Empirical Analysis of Programming Language Adoption. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA, pages 1–18. ACM, 2013. doi:10.1145/2509136.2509515.

[Ora15] Oracle. Guide to Scaling Web Databases with MySQL Cluster. White Paper, April 2015. URL: https://www.mysql.com/why-mysql/white-papers/guide-to-scaling-web-databases-with-mysql-cluster/.

[Par08] Terence Parr. ANTLR—ANother Tool for Language Recognition. Toolkit website, 2008. URL: http://antlr.org.

[Pen04] David M. Pennock. A dynamic pari-mutuel market for hedging, wagering, and information aggregation. In *Proceedings of the 5th ACM Conference on Electronic Commerce*, EC '04, pages 170–179, New York, NY, USA, 2004. ACM. URL: http://doi.acm.org/10.1145/988772.988799, doi:10.1145/988772.988799.

[Qve10] Nikolai Qveflander. *Pushing real time data using HTML5 Web Sockets*. PhD thesis, Ume University, 2010. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.175.2195.

[RBR11] Nicolas Ruflin, Helmar Burhkart, and Sven Rizotti. Social-data storage-systems. In *DBSocial '11 Databases and Social Networks*, pages 7–12. ACM, 2011. doi:10.1145/1996413.1996415.

[Rek92] J. Rekers. *Parser Generation for Interactive Environments*. PhD thesis, University of Amsterdam, 1992. URL: http://homepages.cwi.nl/~paulk/dissertations/Rekers.pdf.

[SJ05] A Vijay Srinivas and D. Janakiram. A model for characterizing the scalability of distributed systems. *SIGOPS Oper. Syst. Rev.*, 39(3):64–71, July 2005. URL: http://doi.acm.org/10.1145/1075395.1075401, doi:10.1145/1075395.1075401.

[SSS+08] Will Sobel, Shanti Subramanyam, Akara Sucharitakul, Jimmy Nguyen, Hubert Wong, Arthur Klepchukov, Sheetal Patil, O. Fox, and David Patterson. Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0, 2008.

[THBG12] Wei-Tek Tsai, Yu Huang, Xiaoying Bai, and J. Gao. Scalable architectures for saas. In *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2012 15th IEEE International Symposium on*, pages 112–117, April 2012. doi:10.1109/ISORCW.2012.44.

[vW65] Adriaan van Wijngaarden. Orthogonal Design and Description of a Formal Language. MR 76, SMC, October 1965. URL: http://www.fh-jena.de/~kleine/history/languages/VanWijngaarden-MR76.pdf.

[WG06] Charles Weinstock and John Goodenough. On system scalability. Technical Report CMU/SEI-2006-TN-012, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2006. URL: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=7887.

[WS04] Lloyd G. Williams and Connie U. Smith. Web Application Scalability: A Model-Based Approach. *Software Engineering Research and Performance Engineering Services*, 2004. URL: http://www.perfeng.com/papers/scale04.pdf.

[Zay13] Vadim Zaytsev. Micropatterns in Grammars. In Martin Erwig, Richard F. Paige, and Eric Van Wyk, editors, *Proceedings of the Sixth International Conference on Software Language Engineering (SLE 2013)*, volume 8225 of *LNCS*, pages 117–136, Switzerland, October 2013. Springer International Publishing. doi:10.1007/978-3-319-02654-1_7.