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

Master's Project in Software Engineering

Lukasz Harezlak
lukasz.harezlak@gmail.com

Summer 2015, 33 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

Software scalability. Cloud scalability. Why is it important Importance of measuring in the clouds and testing optimal architectures.

1.1 Initial Study

What we found out researching.

1.2 Problem Statement

what's the best architecture that answers that best: how to deliver data to geographically distributed users with minimal, consistent and manageable latency how to react scales up and down in response to demand changes (WEBSOCKETS - DISCONNECT SESSIONS? WHEN? This provides some additional challenges when it comes to websockets, since in order to stop an instance one needs to make sure it does not serve any sessions. [MMPROJ9]) according to the models based on the set of metrics described in detail later **??** Also, there is an inherent limitation for scaling a websocket application - a number of TCP ports (and file descriptors available to a ws/wss connections) on a server instance. Hardware limitations are different in an http-based application.

1.2.1 Research Questions

1. Does architecture with geographically aware scaling case can produce better results than the baseline architecture?

2. What is the preferred architecture decomposition for a system with given characteristics - stateless, deployed in the cloud, dynamically scaled up and down, with persistent connections, clients distributed globally and uncacheable data generated in real time?

1.2.2 Solution Outline

How our solution works in short.

1.2.3 Research Method

Software engineering is a relatively difficult field to investigate in terms of lack of clarity on how to do it. Most of the problems are of design, rather than pure scientific, nature. West Churchman coined a specific term for these kind of modern problems - wicked (since they are resistant to resolutions) [MMPROJ1]. Eastbrook et al claim it is often difficult to identify the true underlying nature of the research problem and thus the best method to research it [MMPROJ2]. In their work, they name and compare five most 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?), and descriptive-comparative questions(How does X offer from Y?). The questions of this research are of the last type. The authors suggest pinpointing up-front what will be accepted as a valid answer to the research question [MMPROJ2]. The detailed description of each of the methods helps me to settle for the controlled experiment. Its well-suited for testing a hypothesis where manipulating independent variables has an effect of dependent ones, which is exactly the case of me research. The manipulated variable is architecture decomposition, and the measured ones are determined by scalability measurement models described below.

Research Difficulty

The experiments will be performed in a shared cloud environment, which is inherently unpredictable and changing. Designing and executing a test yielding statistically relevant results where all the necessary variables are controlled (within reasonable boundaries) will be a challenge. 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. Some of the necessary development and data collection might be in relatively low-level technology. The geo-location with reasonable accuracy of the clients might prove difficult too. Overall, its a complex project requiring knowledge of both hardware (protocol), software (architecture) and consisting of multiple disciplines. even linking order in the compiler can change the performance!!! [one of CRIMES]

1.2.4 Hypothesis

The correct geographical decomposition of application stack can lead to vastly improved performance in comparison with baseline architecure(TODO:S????).

1.3 Contributions

Case study on AWS. Architecture decomposition analysis. Deliverable piece of code - metrics framework and auto-scaling scripts.

1.4 Related Work

The big paper on measuring scalability in the clouds. custom routing between regions. A lot on page 14/21 MMPROJ. The big paper on deploying different scalability structures. Small mentions of the others.

1.5 Outline

Here we outline the structure of the thesis. A short paragraph on what happens in each chapter.

Chapter 2

References TO MOVE TO THESIS.BIB

[Master's Project Plan]

[MMPROJ1] Churchman, C. West, "Wicked Problems", Management Science 14 (4) (December 1967). [MMPROJ2] S.Easterbrook, J.Singer, M.A. Storey, D. Damian, Selecting Empirical Methods for Software Engineering Research, Java Report 3 (7) (1998) 5156. [MMPROJ9] N. Grozev, R. Buyya, Multi-Cloud Provisioning and Load Distribution for Three-Tier Applications, ACM Transactions on Autonomous and Adaptive Systems (TAAS), Vol.9 Iss.3, Article No. 13 (October 2014) [MMPROJ10] T. Leighton, Improving performance of the internet, Communications of the ACM - Inspiring Women in Computing, Vol. 52, Iss. 2, p. 44-51 (February 2009)

Chapter 3

Background

3.1 Scalability of Software Systems

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 similar 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).

failures: address space exceeded, memory overloaded, available network bandwith exceeded, internal table filled.

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

failure: resource is overloaded or exhausted and adding capac- ity to the resource does not result in a commensurate ability to handle significant additional demand. For example, adding a processor may not allow a system to meet the additional de- mand if adding the processor also increases overhead significantly.

Scalability is generally desired in the software systems, yet it comes at a cost associated with the system's design. Design of scalable systems is more complex (since there are additional problems that need to be dealt with).

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 smallerchunks. This division happens on different application layers, and different parts of the system now have to communicate and synchronize with each other.

The user traffic, and with it the need for system services and resources, rarely stays constant. From this, a need to be able to scale up and down dynamically in response to traffic arises.

tradeoffs: performance and scalability, cost and scalability, operability and scalability, usability and scalability, data consistency and scalability [WG06] (look there for details)

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 were introduced: first long polling, to enhance the process and reduce overhead. 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.

Before the rise of the websocket protocol, different techniques were used to improve communication between server to client. Among them, the following can be listed (some of them serving different purposes): ajax - a request / response model; was an improvement since the page didnt have to be refreshed anymore to get new data, short polling - using a timer to regulate sending requests to server, similar to refreshing the page, useful when data doesnt change too often, long polling - now considered to be a workaround of preventing creating connections for each request, keeps a connection artificially alive for some time, clients have to reconnect periodically, webRTC - a peer to peer connection server-side events - only allow for a server-initiated communication.

There exist multiple scalability vectors - applications can be scaled in a multitude of different ways. Different decompositions of application stack can be applied. Database layer itself can be scaled up (or out) in numerous ways. They and their benefits are described below.

The two researchers present us with an interesting thought - software engineers need to design for the cloud, not only to deploy in it. To facilitate that process, they propose an adaptive dynamic provisioning and autonomous workload redirection algorithms. [MMPROJ9]

3.1.1 Existing Approaches

A few traditional approaches of tackling an issue like that exist already. Among them we can enumerate: Scaling up (one massive instance), end-user experience is at the mercy of the unreliable Internet and its middle-mile bottlenecks Traffic levels fluctuate tremendously, so the need to provision for peak traffic levels means that expensive infrastructure will sit underutilized most of the time model does not provide the flexibility to handle unexpected surges [All 28] This will be selected as the baseline architecture - first with isolating the network effects (tested on an internal network), and then in the real-life cloud scenario. This way the real impact on the performance of the architecture of network variables can be established. Scaling out randomly - firing up new system nodes in new data centres, Averaging out geographical locations and firing up instances 'in the middle', Content Delivery Networks - these only handle static assets. Websockets do not support CDNs, because the WebSocket protocol is stateful. Big Data Center CDNs potential improvements are limited because the CDNs servers are still far away from most users and still deliver content from the wrong side of the middle-mile bottlenecks Highly Distributed CDNs putting servers within end-user ISPs Algorithmically predicting message frequency and opening / closing persistent connections according to this. Peer to peer networks. [MMPROJ10] Scaling up (using a bigger server instance) is easy to implement, but costly, even extremely when you start pushing at current hardware limits. Which, with a websocket-based application is not an impossible option. Instead, one can perform horizontal scaling out and cost of hardware can be reduced dramatically this way. He discusses different types of balancing: application layer balancing, business load balancing, and anticipating load. He claims that the overhead of parsing requests in the application layer is high thus limiting scalability compared to load balancing in the transport layer. [MMPROJ14] client state needs to be stored on the server but in a layer shared between the webservers - the more of it, the bigger the latency When the client reconnects, the server calculates a delta, the difference between the current market state and the last values sent to the client and only sends those values that have changed. [MMPROJ14] The overhead of parsing requests in the application layer is high thus limiting scalability compared to load balancing in the transport layer.

3.1.2 Cloud Scalability

General.

3.1.3 Data-layer Scalability

Data layer is often a performance bottleneck because of requirements for transactional access and atomicity - it is hard to scale out because of this [MMPROJ9].

The technology stack I am working with in this scope consists of mysql as a persistent storage and redis as a key-value cache. Described in details 4.3.

Many solutions and strategies for dealing with database scalability have emerged, including nosql

and newsql databases, data replication, caching and database sharding [MMPROJ8]. In Amazons environment, Grozev and Buyya suggest using Elasticache service [MMPROJ9]. We cannot use this, also described in 5.4.

Cooper et al touch on that subject in their work [MMPROJ7]. They claim that in scaling out one should aim for elasticity and high availability. These are hard to achieve using traditional database systems. They emphasize that there are protocols that help achieve strong transaction: two-phase commit and paxos. They also give an overview of different database systems, including PNUTS, BigTable, HBase, Cassandra, Sharded MySQL, Azure, CouchDB, and SimpleDB. Mysql they use does not support elastic growth and dynamic data repartitioning - Sharded MySQL is inherently inelastic. In their work [MMPROJ7], we can find enumeration of classic data-related scalability tradeoffs, such as latency and durability. They talk about different replication techniques - synchronous and asynchronous and different data partitioning solutions.

Mysql White paper [MMPROJ19] gives us a good overview of how scalability works in Mysqls case. Authors suggest identify which characteristic the application falls into - lots of write operations, real-time user experience, 24 x 7 user experience or agility and ease-of-use. My application falls into the second tier. They claim to support (among others) auto-sharding for write-scalability, active / active geographic replication and online scaling and schema upgrades. Geographic replications offers distribution of clusters across remote data centers, which they claim helps reduce latency (important in our case). Authors show how mysql is optimized for real-timeness: data structures are optimized for in-memory access, persistence run in background processes, all indexed columns are stored in memory. They claim that mysql clustered is very well-suited for on-line, on-demand scaling, which provides a contrast to what cooper et al have found (careful here, son) [MMPROJ7].

Liu in his Eventual Consistency [MMPROJ20] writes about BASE property - basically available, soft state, eventually consistent. Its a complement of ACID. Author claims we lack precise metrics to measure its aspects and thats why every implementation implements eventual consistency differently. He writes about different implementations. Most importantly for my research, he explains mysql cluster implementation - that it performs much like an ACID database but with the performance benefits of a cluster and can be configured at multiple topologies, not only the basic master-slave. Consistency differs per configuration.

Ruflin et al in Social-Data Storage-Systems [21] evaluate different storage system types: rdbms, key-value, column store, document store and graph databases. Two first are of my interest, especially since they evaluate technologies I use - redis and mysql. The authors mention unconventional usages of Mysql, as a key value store e.g. by Twitter. Reddit also uses it, with a single table [SOURCE]

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 [21].

Redis is said to be of limited scalability, because of the fact that for good performance the whole data set should fit into memory.

Further improvements to mysql: Pakkonen and Pakkala show how MySQL performance can be increased [22], e.g. by disabling automatic committing or binary logging (binlog) output. They used the already mentioned YCSB benchmarking tool [7]. They also present the results of the tests they performed - switching off auto committing changes the latency from (1.4s to 130 ms) on their data sample, which is a huge improvement.

3.1.4 Websocket Scalability

A nice introduction into push-base communication over the internet can be found in Agarwals work - Toward a Push-Scalable Global Internet [MMPROJ11]. The key message in this paper 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 multiple times a minute. Author here proposes a content-based, machine-learning optimized solution for closing and opening connections when needed. The result he achieves is that the number of hours of active always-on connections can be cut by half while still

achieving real-time message delivery for up to 90 percent of all messages.

The author performs the following reasoning in his study. 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.

my own research has shown - difficult, one needs to tweak kernels, cloud scalability built for http

previous: ajax, long polling, short polling, websocket [MMPROJ11]

M. Franklin and S. Zdonik provide an insight into the history of push-based technologies. Their paper from 1998 [MMPROJ12] classifies communication mechanisms into aperiodic pull, periodic pull, aperiodic push and periodic push.

Cassetti and Luz [MMPROJ15] claim that overhead introduced by the websocket protocol and WebSocket API is rather small as compared to other communication methods. Furthermore, you can achieve superior bandwidth and performance when using websockets.

3.1.5 Geographical Distribution

The topic is nicely introduced by Tom Leighton in his article Improving performance of the internet [10]. 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 local ¡ 100 miles 1ms regional 500 - 1000 miles 15ms cross-continent 3000 miles 50ms multi-continent 6000 miles 100ms

The longer data must travel through the middle mile, the more it is subject to congestion, packet loss, and poor performance. This is why the company he works for, Akamai, locates servers close to smaller network, end users (on a scale that I cannot reproduce in this research - requires control over infrastructure, in this case it's Amazon's). He claims that big websites have at least two geographically dispersed mirror locations to improve performance, reliability and scalability [MMPROJ10]. In his work, Leighton includes a set of guidelines to follow when thinking about geographical scalability: reduce transport layer overhead (this is achieved partly in my case by using websockets), prefetch embedded content, assemble pages at the edge, offload computations to the edge, ensure significant redundancy in all systems to facilitate failover, software logic to provide message reliability, distributed control or coordination.

Ed Howorka in his interesting paper Colocation beats the speed of light [MMPROJ13] 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 my case) can be replaced by colocated servers (located next to, in my case, users). What is more, he implies that server colocation provides a better solution for high-speed applications (as opposed to using a big, centralized server located in the middle). His formal proof in the article (why isn't this a case for us).

3.1.6 Measuring Scalability

What we have from literature study. ONLY THE GENERAL STUFF - WE HAVE A FULL CHAPTER 6 ON THIS.

3.1.7 Benchmarking

[MMPROJ] important to have random distributions(uniform, zipfian, latest, multinomial)

Benchmarking Crimes

3.1.8 Websocket Protocol

websocket draft - RFC6455

[http://stackoverflow.com/questions/4852702/do-html-websockets-maintain-an-open-connection-for-each-client-does-this-scale/2534022025340220] 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. http connections more expensive: - Each HTTP connection carries a lot of baggage that isn't used most of the time: cookies, content type, conetent length, user-agent, server id, date, last-modified, etc. Once a WebSockets connection is established, only the data required by the application needs to be sent back and forth. - TYPICALLY: HTTP servers are configured to log the start and completion of every HTTP request taking up disk and CPU time. It will become standard to log the start and completion of WebSockets data, but while the WebSockets connection doing duplex transfer there won't be any additional logging overhead (except by the application/service if it is designed to do so) - interactive applications that use AJAX either continuously poll or use some sort of long-poll mechanism. WebSockets is a much cleaner (and lower resource) way of doing a more event'd model where the server and client notify each other when they have something to report over the existing connection. obviously don't support cdn

As for decreased latency via WS vs. HTTP, it's true since there's no more parsing of HTTP headers beyond the initial WS handshake. Plus, as more and more packets are successfully sent, the TCP congestion window widens, effectively reducing the RTT.

Think of it this way: what is cheaper, keeping an open connection, or opening a new connection for every request (with the negotiation overhead of doing so, remember it's TCP.)

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.

Chapter 4

The System Under Test

real time prediction parmet parimutuel pools directed contracts sort of a stock market for real-time prediction on anything, e.g. live sport

4.1 General Purpose of The System

first applicatoin - live sports low manageable latencies important

4.1.1 Sample Use Case

A good example e.g. can be predicting the outcomes of certain drives in the football match as they happen. A sample case: Manchester United - Liverpool live game, live data coming from UK servers, introduced into the system through a UK node, most users (who also generate data that needs to be distributed) in China (10k), India(10k), Australia(3k). Where should websocket servers which distribute messages spun up for minimal latencies for all clients? Is it faster to spin up db replicas on that nodes too?

4.1.2 Users of the System

in stadium on couch dekstop tablet mobile Users of the platform are mostly on mobile networks (that often drop), so reconnecting them quick to the right (providing lowest latencies) instance is important. lumpy demand - it comes in 2-hour-long spikes and then can go quiet for days

4.2 Basic Architecture

4.3 Technology Stack

Python + redis + javascript + mysql (Ampersand) cloud stack described here: 5.4

4.4 Unique Aspects of The System

critical data flows over websockets Users receiving data shared locally should receive it at the same time as data shared globally (with as low a latency as possible). This creates a need for globally consistent and manageable latency between end user and the system. Connected users and sources of data are geographically changing. Users geographical center of mass is changing for each peak of demand. Extremely lumpy demand (peaks lasting around 2 hours). This creates a need for being able to quickly scale up and scale down. New data is generated every few seconds by the users so caching the content and distributing geographically 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 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

5.2.1 Kernel tuning

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.

look for $_{w}hatcanlimitconcurrentsockets?_{i}nlabnotes$

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.htmlmode-massconnect

Understanding tcp sockets: sockets are left in $\text{TIME}_{W}AITontheserverwhentheclientsuiteiskilledwhichmakessense$ $delaytime$)

5.3 Baseline architecture on a local network

Architecture diagrams. Technical details - local server capabilities. Load testing.

5.4 Cloud architecture setup

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 http://docs.aws.amazon.com/Route53/latest autoscaling only within one region, multiple availability zones http://docs.aws.amazon.com/AutoScaling/latest/Develop http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/how-as-works.htmlarch-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, $\text{INSSUFICIENT}_{D}ATA$), $cantriggerscalingup/downpolicyonanautoscalinggrouppolicyvariants(changeExactCapacit$ $//docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/as-scale-based-on-demand.htmlhttp:$ $//docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/AlarmThatSendsEmail.htmlhttp:$ $//docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/US_{A}larmAtThresholdEC2.html$

load balancers with AutoScaleGroups. vs what? instances behind each groups. automatically scalable, connected to external redis and mysql instances. problems with rds and elasticache

5.4.1 Baseline architecture deployed in the cloud

5.4.2 Improved architecture deployed in the cloud

5.5 Experiment deliverables

Chapter 6

Scalability Measurements

The previous description 3.1.6 concerned general stuff. 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]

6.1 Selected metrics

latency (messaging + handhsaking) sustainable concurrent websocket connectoins infrastructure cost (per unit of time, supporting the same number of users) architecture reaction speed to changes in demand (scaling up and scaling down) dropped connections cpu usage memory usage network in network out iops reads + writes CRAM METRICS TODO: review

6.1.1 EC2 Available metrics

collectible every minute (in detailed mode), by default every 5 minute, paid additionaly 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$_c$*oncepts.html*]*custompo*
$min, max, sum, avg, sampleCount(countofdatapointsbutresearchthat) cpuutilization network in/out(inbytes) diskrea$
$bytes) status checks$

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.

6.2 Selected model

[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.

6.3 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?

6.3.1 Load Testing

Described [**?**]

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[MMPROJ7]. Another distribution is suggested by Grozev and Buyya [MMPROJ9] - Poisson distribution with a constant mean.

6.4 Baseline, Deployed in the cloud

6.5 Improved, Deployed in the cloud

Chapter 7

Experiment results

Chapter 8

Evaluation

8.1 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

Conclusion

Chapter 10

Further work

Chapter 11

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

Chapter 12

Front Matter

The first thing is to connect the class by saying:

```
\documentclass{uvamscse}
```

12.1 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.

12.2 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.

12.3 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.}
```

12.4 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.

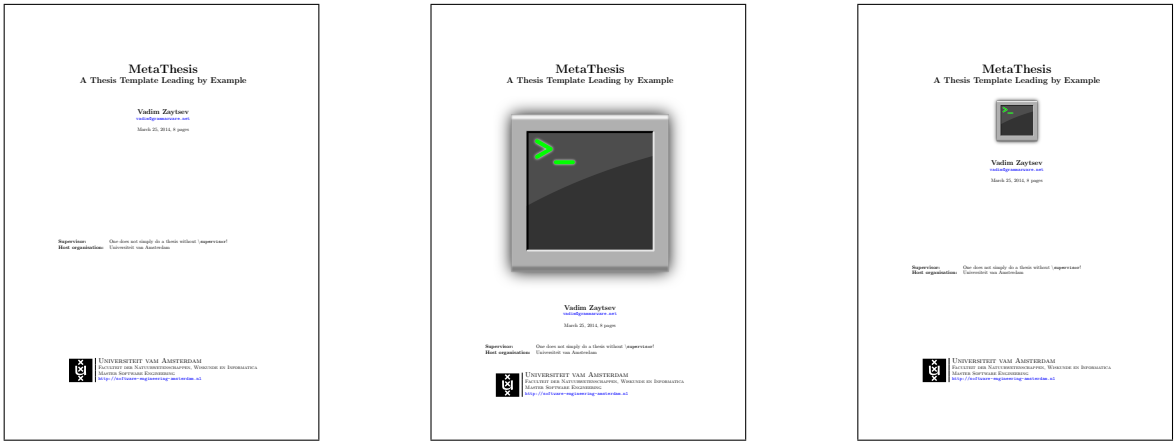

Figure 12.1: A hypothetical thesis title page without a cover picture (on the left), with an overly large one (in the centre) and with a tiny pic (on the right).

12.5 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 12.1.

12.6 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 13

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.

13.1 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 § 14 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.

13.2 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.

13.3 LaTeX details

13.3.1 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

13.4 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 13.1: 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 13.2: 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 13.3: 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 13.4: Code in Python

Chapter 14

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.

14.1 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,
}
```

14.2 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",
}
```

14.3 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",
}
```

14.4 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.

14.5 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",
}
```

14.6 Wikipedia

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

14.7 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

[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.

[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.

[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.

[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.

[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.

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

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

[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.