BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

PERFORMANCE TESTING AND ANALYSIS OF QPID DISPATCH ROUTER

TESTOVÁNÍ A ANALÝZA VÝKONNOSTI QPID DISPATCH ROUTERU

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. JAKUB STEJSKAL**
AUTOR PRÁCE

SUPERVISOR **Ing. TOMÁŠ FIEDOR,**
VEDOUCÍ PRÁCE

BRNO 2017

Abstract

[[Doplnit]]

Abstrakt

[[Doplnit]]

Keywords

Testing, performance analysis, network technologies, router, Qpid-Dispatch

Klíčová slova

Testování, analýza výkonu, síťové technologie, router, Qpid-Dispatch

Reference

STEJSKAL, Jakub. *Performance Testing and Analysis of Qpid Dispatch Router*. Brno, 2017. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Tomáš Fiedor,

Performance Testing and Analysis of Qpid Dispatch Router

Declaration

Hereby I declare that this bachelor's thesis was prepared as an original author's work under the supervision of Mr. X The supplementary information was provided by Mr. Y All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references. [[**dopsat**]]

. . . . . .
Jakub Stejskal
December 2, 2017

Acknowledgements

Here it is possible to express thanks to the supervisor and to the people which provided professional help (external submitter, consultant, etc.).

Contents

Chapter 1

Introduction

Good application performance is one of the main goals during the software development. But what makes software performance so important? Software reliability has to be guaranteed by the owner, but with undesirable performance there could be a lot of issues. Thus can badly influence software behavior. And this can cause a significant outflow of the consumers, and even brand destruction, financial damage, or loss of trust. These few reasons should be enough to do a proper performance testing before every software release, especially for large projects where industries guarantee certain level of software behavior and they would not be able to assure it with insufficient performance testing. Great emphasis on software performance is, in particular, in space programs, medical facilities, army systems or energy distribution systems. In these fields it is necessary to ensure proper application behavior for a long time under high load and without any unexpected behavior such as high response time, frequent delays or timeouts, because every failure is paid dearly.

Nowadays every developer should try to use well established frameworks which can make theirs work easier. Frameworks handle complex underlying issues such as security, performance, and code clarity. This way developers can invest more time in the actual functionality and meet the application requirements, since frameworks are usually optimized for one particular job. In the past every developer had to spent significant portion of development time tuning performance which led to spending more time and money for software development. But not everyone has enough knowledge of performance testing and this makes performance analysis and optimization even more difficult. This leads to a need for specialized performance tools which can provide more sophisticated information, however, actually useful tools are usually proprietary and/or too expensive.

A very important part of the performance analysis is the right choice of *key performance indicators* (KPIs) [14] and effective interpretation of the results. The right choice of KPIs allows faster detection of performance problems and help developers with fixes and meeting the *performance standards* [14] set up by application owner or customer in time before the release.

In general an application performance is important. However, smooth network application or hardware performance became much more demanded nowadays, since most of the communication is via the Internet. Obviously when you make a payment in your internet banking you definitely want to have a stable connection to your bank's website without any delay. Network stability is significantly influenced by network components like routers and switches and hence their performance should be under utmost case. We refer to network performance testing as measurement of network service quality which is directly influenced by *bandwidth, throughput, latency*, etc.

For performance testing of particular network messaging system developed by *Red Hat Inc.* there is an existing solution —Messaging Performance Tool (MPT) [15]. MPT is specialized for the performance testing of *Broker service* [16] — network application level software cooperating with *Qpid-dispatch service* [17] in the network as the message distributor. Unfortunately, the current version of MPT does not support performance testing of enough component like the Router component, Qpid-dispatch. In this work we will focus on this particular short coming and develop a worthy solution allowing proper performance testing of the Qpid-dispatch service.

This thesis is structured as follows. First, we define fundamentals of performance testing in Chapter 2. The rest of the thesis focuses on performance testing and analysis of Qpid-dispatch, an application level router designed by Red Hat Inc. Qpid-dispatch performance testing is based on MPT described in Chapter 3. Description includes *measurement process* and *measured data description and evaluation.*The main goal of the thesis is to analyze MPT and design module for the Qpid-dispatch performance testing as described in Chapter 4 together with used protocols and *Automatic Topology Generator* for semi-automated network generation and deployment. Used technologies, tools and implementation processes of each component are described in Chapter 5. The most important part of the thesis is Chapter 6, containing the data gathering from routers located in different types of topology, data evaluation and representation which leads to conclusion about performance of Qpid-dispatch. Finally, Chapter 8 summarizes the thesis and proposes ideas for future use of developed tool.

Chapter 2

Fundamentals of Software Performance Testing

Usual goal of performance testing is to ensure that the application runs reasonably fast enough to keep the attention of users, even with unexpected amount of clients using the application at the same time. But why is it so important to have the application optimized for the best speed? Simply, when your application have slow response, long load time or bad scalability, the first website which user will visit afterwards will be web of your competitor. That is the reason why speed is currently one of the most significant performance factor of common performance problems. This chapter summarizes the fundamentals of performance testing which includes common performance processes, issues, and metrics, based on knowledge available in [14, 13, 8, 1].

2.1 Performance Testing Process

The main goal of the performance testing is to ensure the following application attributes [9]:

Reliability and Stability — the ability of software to perform its functions in system environment under some system load for acceptable period of time,

Scalability — the ability of software to behave properly under various types of system load and handle increasing amount of workload (such as network traffic, server load, data transfer, etc.),

Processing time and Speed — the ability of software to react quickly without low response time during any acceptable system load,

Availability — the ability of software to make all of its functions available during any acceptable system load.

Similarly to software development process, performance testing process consist of usual engineering steps ranging from requirements definition to data evaluation. These steps also includes design, implementation of performance tests and execution with data collection. The graphical representation of the performance testing process is depicted in the Figure 2.1.

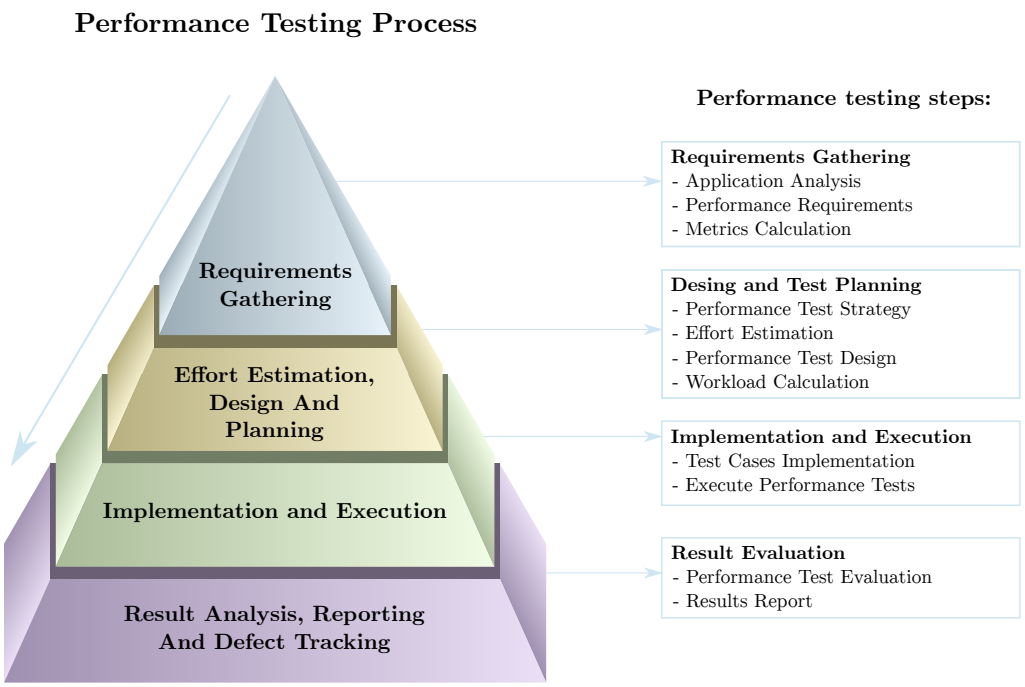

Figure 2.1: Performance Testing Process with the four most important parts and theirs individual steps based on [18].

The first step of performance testing process is the selection of *performance requirements* for the application. In this step, testing engineer has to analyze *software under test - SUT*, suitable performance metrics, that will model the application performance, and set performance requirements. The result should include answers to questions such as:

- How many end users will the application need to handle at release, after 6 months or in 1 year ?
- Where will these users be physically located, and how will they connect to the application?
- How many end users will be concurrently connected in average at release, after 6 months and 1 year?

Based on answer to these studies, the engineer should be able to select important key performance indicators for performance test cases. Some of these indicators may be *response time*, *stability*, *scalability*, or *speed*. However, there is huge amount of possible indicators so it is necessary to properly analyze the whole application and also take into consideration another needs like an error rate, system resources, etc. Result of this phase should be a binding document with all performance requirements to be tested and, in case of detected performance degradation, such defect must be fixed with reference to this document.

The next step is to define the *performance testing strategy*, corresponding to planning and design. It is extremely important to allocate enough time for SUT testing effectively, because, as it was mentioned in Chapter 1, performance testing is not an easy task and detecting all of the possible issues of tested components is very time consuming process. Every plan should take into account the following considerations:

Prepare the test environment — this step include choosing hardware machines for testing and installing the necessary software for running load injectors, tested components, etc.

Provide sufficient workload injectors — preparing the workload injector may take few days; we usually requires a few workstations or servers to simulate real traffic.

Identify and implement use cases — this include identification of important parts of the system which may have an impact on performance; time needed for each use case may be different because some use cases can be simple such as navigating to a web application home page, but some may be complex such as filtering specific communication.

Instrument the test environment — install and configure the monitoring software on the test environment.

Deal with detected problems — test can detect significant performance issues and their fix may take a long time.

While this process seems trivial, the opposite is true, in particular in case of network applications. Most of performance issues manifest at big workloads or high number of users, e.g. when million users are sending requests to the network device at the same time it could lead to an unacceptable device crash. Workload injectors are designated to simulate real user activity, and allows automatic analysis of performance behavior for tested application or device. Depending on the used technology, there can be a limit on the number of virtual users that can be generated by a single injector. These automated workload injectors are necessary for effective performance testing.

After describing the plan we implement and execute proposed test cases. Environment and workload injectors are ready for execution, so last step before the testing itself is the implementation of tests. Thanks to the careful planing, engineers should have enough time to implement test cases with reference to proposed design.

Final step of performance testing process is results evaluation. Output of this step is usually technical report with all selected performance key indicators, used workload and gathered data for each test case. Then follows the data evaluation with thorough analysis of degradation localization. Additionally, the report usually contains syntactical graphs which display performance metrics along the duration of test execution.

2.2 Performance Issues

Performance issue is a common label for an unexpected application or device behavior which affects its performance. Usually, those issues are hard to detect because they manifest only under certain circumstances such as high load or long application run time. In the network applications there are several particular issues that are more frequently occurring than others. In following, we will describe selected issues in more detail.

Performance Degradation

Unclean code usually leads to inefficient algorithms, application deadlocks, or memory leaks, which all can eventually cause the performance degradation. The problem is that these issues are usually detected only during the long run time of application or inability of an application to handle high load. For this kind of issues there is a performance testing method called *soak testing* [7, 11] which is described in Section 2.3. Soak test is intended to identify problems that may appear only after long period of application run-time, hence its necessary to run this type of tests during network application development. The network applications are usually need to be available for 24 hours per day. The duration of a soak test should have some correlation to the operational mode of the system under test. Following scenarios may represent performance issues detectable by soak tests:

- a constant degradation in response time, when the system is run over the time,
- any degradation in system resources that are not apparent during short runs but will surface during long run time such as free disk space, memory consumption or machines handles,
- a periodical process that may affect the performance of the system, but can be detected only during long run time as backup process, exporting of data to a 3rd party system, etc.,
- development of new features for already using components.

Response Time

Response time is time it takes system to accept, evaluate and respond to the user for his request e.g. HTTP request for particular website. Different actions and requests can have significantly different response time and with that provide different load on the system. For example retrieving document from web-server by its ID is considerably faster than searching for the same document by keywords. Response time is mostly measured during the *load test* [11] of the application. Well designed test should consider different types of load on the system, various kind of requests, and different number of connected end-users at the same time. For user based systems we usually consider 3 threshold for the response time values:

0.1 second — this represent an ideal response time for the application, because user feels that system is reacting instantly and does not notice any interruptions.

1 second — this is the highest acceptable response time when user still does not feel any interruptions, but can feel a little delay; this still represent no bad impact on the user experience.

10 second — this is the limit after which response time become unacceptable and user will probably stop using your application.

However response time limits for messaging system are more strict. They could acquire values in milliseconds or less.

[[Prvni iterace]]

Traffic Spikes

As *traffic spike* [13, 4] we can understand the sudden degradation of one of the performance metrics such as *throughput, bandwidth, error rate, response time* but also resource usage such as *memory usage, disk space, etc.* In real network, spikes are result of high workload, e.g. caused by higher amount of users trying to concurrently use the service over the network. For example we can experience sudden traffic spike in response time after publishing new popular viral context on video servers, start of sales events, reservation of limited amount of tickets or subject registration at university.

Traffic spikes can lead to inappropriate system behavior such as *long response time*, *bad throughput* and *limited concurrency*. To prevent the impact of traffic spikes on system performance, it is necessary to do sophisticated infrastructure monitoring and network load analysis, in order to distinguish between normal traffic and attack on the system. Suitable method for testing of spikes is called *stress testing* [11] and it is described in Section 2.3 in more details. Network system should also be scalable, thus it should be able to redirect traffic to another node with same service in case of high load which can cause performance issues due inappropriate resource usage.

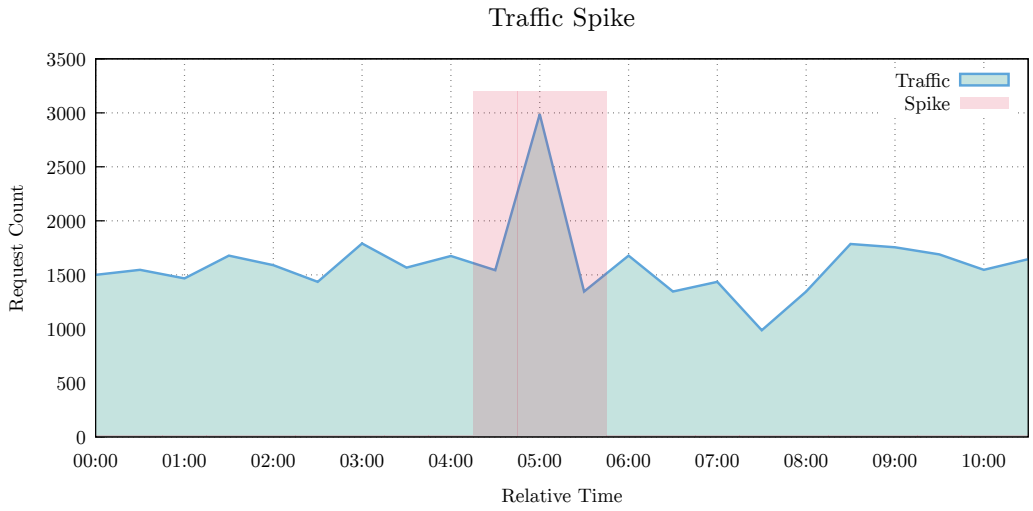

Figure 2.2: The graph shows amount of concurrent sessions depending on time. During to network traffic monitoring the traffic spike occurring around October 11th.

2.3 Types of Performance Testing

For performance testing there are many types of suitable test methods. Which test you should use is determined by the nature of the system, testing requirements or how much time we have left for the performance testing. The following terms are generally well known and used in practice and each of them characterizes category or suite of the tests. Their description is based on knowledge available in [3, 7, 14, 1].

Load Testing

Load testing is a testing method which studies how the system behaves during different types of workload within acceptable time range. Basically it simulates the real-world load.

During the load test we mainly focus on metric response time of the system for requests. Requests are generated by users or another systems communicating with the SUT. Main goal is to determine if the system can handle required workload according to performance requirements. Load test is designed to measure the response time of system transactions under normal or peak workload. When the response time of the system dramatically increases or becomes unstable, we conclude that system reached its maximum operating capacity. After successful testing, we should mark the workload requirements as fulfilling or analyze gathered data and report issues to the developers. In the Figure 2.3 you can see the graph of load test showing workload of raising requests to the web server at the same time where the system response time does not exceed 3.5 seconds.

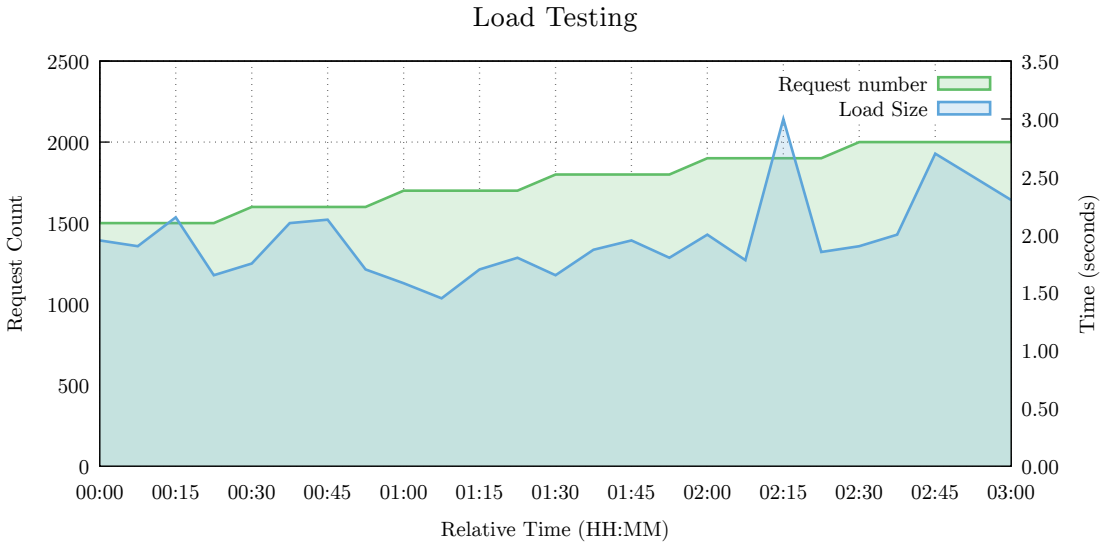

Figure 2.3: Response time of the system during the load testing depended on by load size.

Stress Testing

Stress testing is the specific type of load test, where engineers does not measure normal workload, but focus on unexpected workloads and traffic spikes. The main purpose is to find how the system behaves in extreme conditions such as enormous number of concurrent requests, using a server with much less memory or a weaker CPU and analyze system performance threshold. Its very useful to know performance threshold because you want to know the difference between performance under normal workload and performance threshold when the system is overloaded. The following enumeration collect common stress test scenarios:

- Monitor the system behavior when maximum number of users logged in at the same time.
- All user performing critical operations at the same time.
- All users accessing the same file at the same time.
- Hardware issues such as server in cluster down.

When engineers finish stress testing and found system limits, they also can test the system recovery after crash during finding of the system limits.

In the Figure 2.4 is recorded stress testing with raising load and response time. Everything is fine until the amount of requests exceed 3000. With higher load there comes performance issues which leads to unexpected rise of the response time.

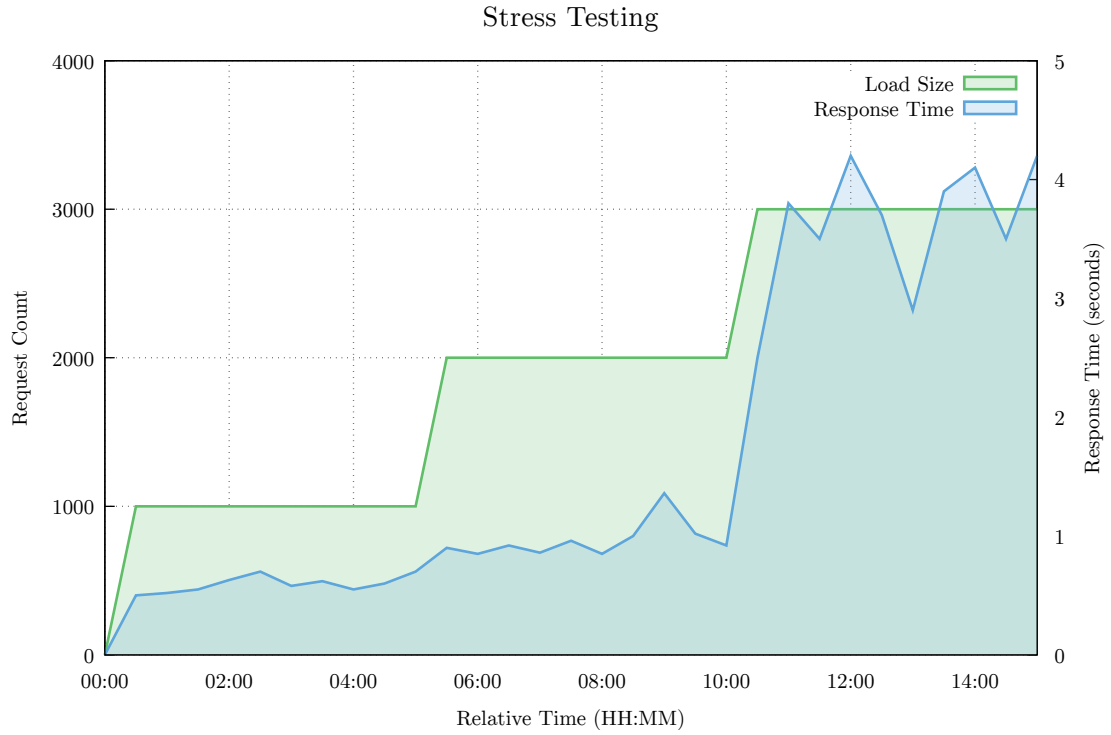

Figure 2.4: Stress testing diagram capturing dependency of response time on mount of requests.

Soak Testing

Soak, or stability/endurance testing refer to method, where goal is to identify problems, that may appear only after extended period of time. The system seems stable for one week, but after longer period than that, problems such as memory leaks or not enough disk space can appear. Primary measured metric during soak tests is memory. The following are common issues found by soak test:

- Serious memory leaks that will eventually result into system crash.
- Improperly closed database connections could starve the system.
- Improperly closed connections between system layers could stall any of the system modules.
- Step-wise degradation could lead to high response time and the system become inefficient.

This sort of test needs for effectively use appropriate monitoring system. Problems detected by soak tests are typically manifested by gradual system slowdown in response time or as sudden lost of system availability.

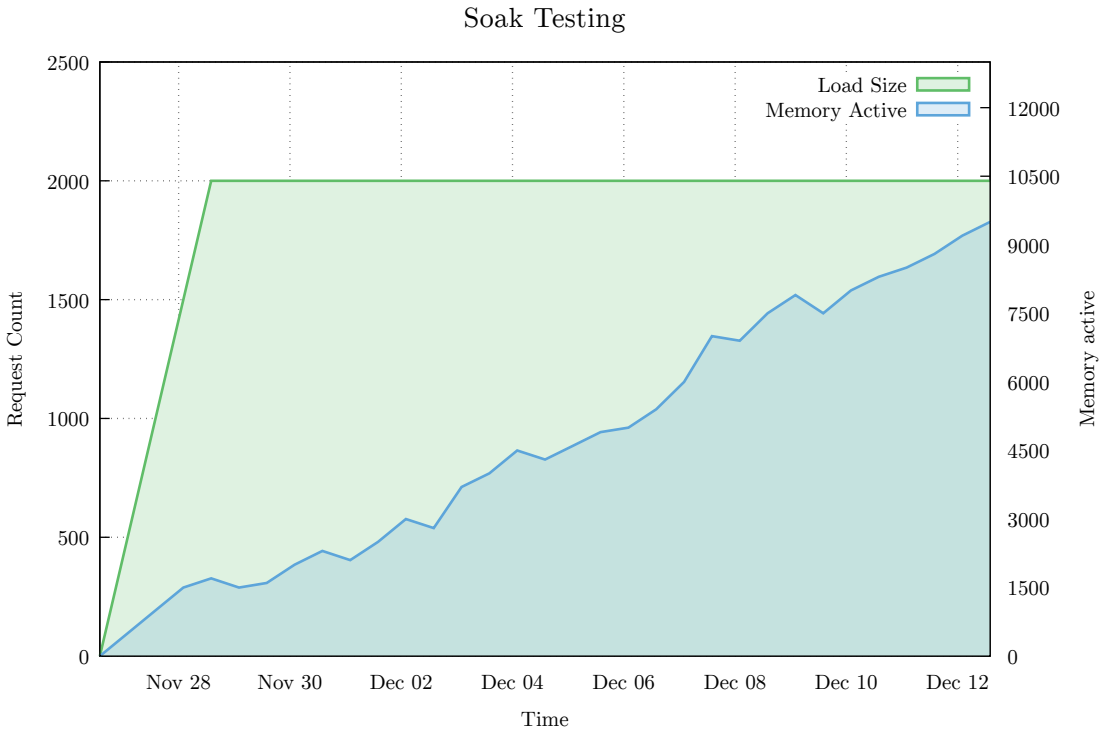

Figure 2.5: Soak testing with memory usage dependent on time.

In the Figure 2.5 you can see raising memory usage after period of time. The STU can handle requests but as time goes by memory usage is to high that the STU will crash. This may be caused by memory leak or an inappropriate algorithm use.

Smoke Testing

Smoke testing method is inspired by similar hardware technique, when engineers checks for the smoke from the device after tuning power on. Basically, its very similar for software, since main goal of smoke test is to test basic functionality of the system and tell to the engineers if the system is ready for build. However, smoke tests are testing functionality on a surface level, so its not enough for testing deeply basic systems functions. When smoke tests fails, the system are tagged as unstable, because its can't ensure basic functionality and its not tested anymore until the smoke test will pass because it should be waste of time. Smoke test are designed to uncovering obvious errors which saves time, money and effort of the engineers. Smoke tests should be used with every new build, since new features could harm previous system functionality.

Regression Testing

With software development comes times, when engineers develop new feature and they want to update previous build and now is the time for regression testing. *Regression tests* [2] are designed to test functionality of the latest build updated with new feature. The main objective is to determine, if new feature affects already functional parts of the system. This type of tests are very important, because engineers do not always know, which part of the

system will be indirectly affected. During regression testing, new test cases are not created, but previous test cases are automatic re-executed and analyzed.

Benchmark Testing

Benchmark testing [12] is method, which is collecting performance data during the system run on different hardware machines. Gathered data have significant value when we want smooth run of the system on an older hardware, hence we can discover performance issues under normal load. However, the system does not run smoothly on prepared hardware, only options is to run benchmark tests on different machines with different hardware and under different load.

2.4 Performance Metrics

During performance testing we can monitoring a lot of metrics, which some are more important than other based on the system purpose. The following metrics are the most common that are monitored during performance testing of all application, no matter of developing language.

[[Doplnit, jak se realne meri metriky vykonu!!!]]

2.4.1 Throughput

Throughput is a metrics, which refer to number of requests per second that the system can handle. In the network, there is *network throughput* that is the rate of successful message deliver over a communication channel. Throughput is usually measured after a warm-up period of time after the commencement of traffic, because it takes a while before the throughput stabilizes **[[WHY?]]**. Throughput is measured by load testing; strategy for measuring throughput is to continuously raise the load until response takes longer that acceptable threshold.

2.4.2 Response Time and Latency

Response time as issue was already mentioned in Subsection 2.2; response time as metric is consist of two parts which are *latency* and *service time*.

Service Time

Service time is time it takes system to evaluate and send response for users request. In particular, when user send request for web page to server, it takes the server time to evaluate request and send proper response back to the user, this is the service time. Measurement can be perform easily with stopwatch which starts at receive request and stops after response send. Service time could be affected by any item which leads to performance degradation as it was described in Subsection 2.2.

Latency

Second part of the response time is latency [6, 5], which is a delay between sending request on the client side and receive it for evaluate on the server side. Hence latency is

common problem in the network systems such as data center, web server, etc., because request/response needs to travel over the physical medium between client and server. Client and server could be located on different continents, thus the message have to travel long distance and latency is increasing.

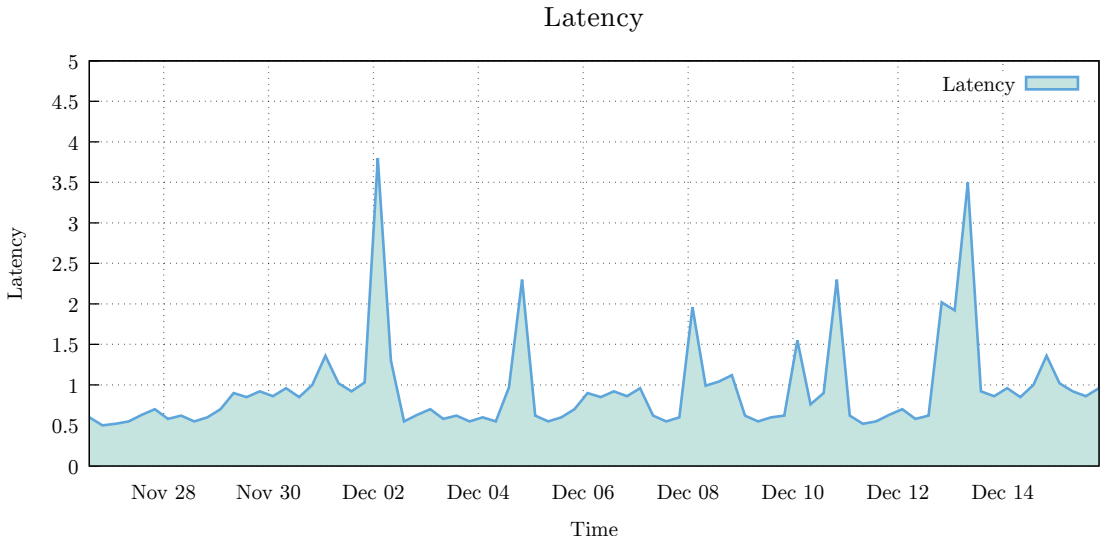

Figure 2.6: Latency diagram with system's response time based on the date.

Average and Percentile Response Time

There are two ways of measuring response time [10]; Average (mean) response time is calculated as sum of all measured time divided by count of users requests. This sounds cool and easy, but many times, average response time does not reflect real response time of the system. How is that possible? In reality, most application have few very heavy outlines such as very slow transactions. In the Figure 2.7 you can see few slow transactions which drag the average of response time to the right. Thus leads to inaccurate determine of response time.

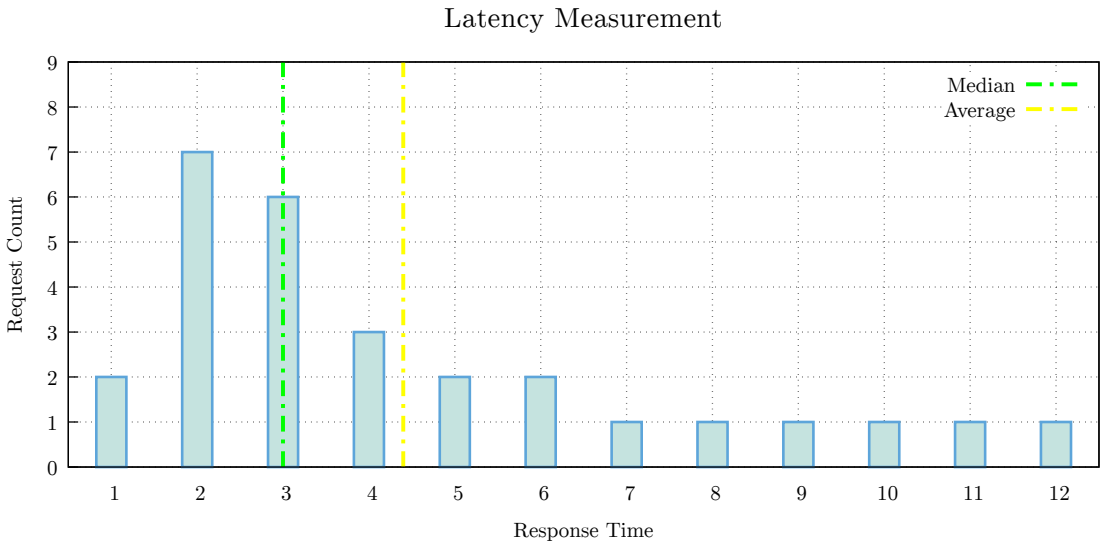

Figure 2.7: Transactions response time with calculated average and median of response time. Average represent inaccurate response time in this case which is higher than real one.

On the other side, better solution how to determine response time is Percentile. In the Figure 2.7 you can see *median* value, which reflect more realistic value of the system response time. In this case, there is no problem, because user will expect slower response time than it has in real. In the figure 2.8 there is different situation. Average response time is better than median, which reflect into expectation of faster system response time than it has.

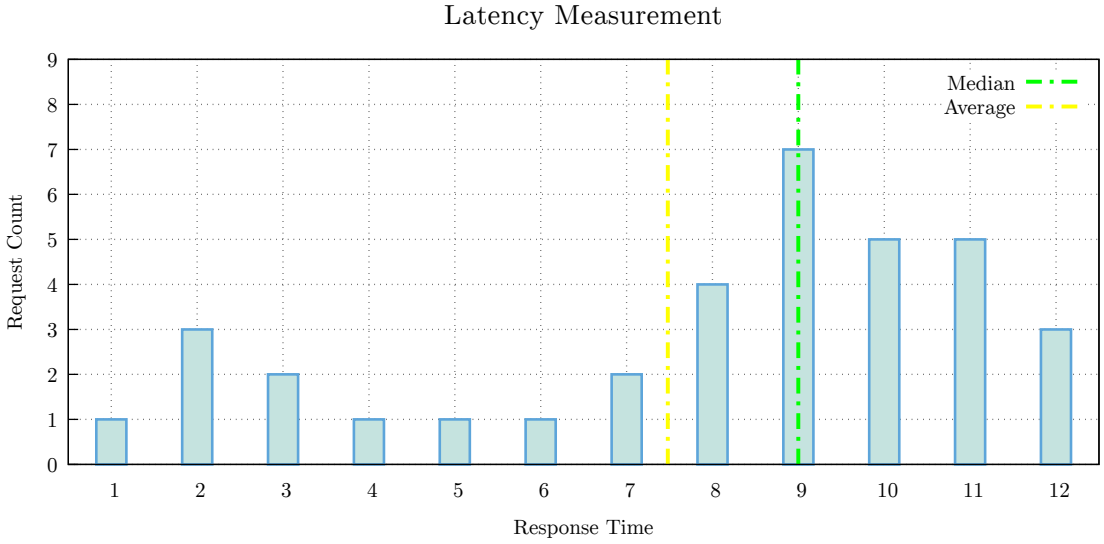

Figure 2.8: Transactions response time with calculated average and median of response time. Average represent inaccurate response time. Average says, that STU is faster than is in reality.

In this case, a considerable percentage of transactions are very fast (first 20 percent), while the bulk of transactions are several times slower. Thus, calculated median gets more realistic values than average response time.

2.4.3 Resource Usage

At long run applications running at servers, there is a limited amount of resource available for use. Thus makes resource usage another important metric, which needs to be monitored since not enough resources could shut down the whole system. Main resources for monitoring and utilization are:

CPU usage — inappropriate using of CPU could lead to performance degradation, because low priority processes may occupy CPU ahead of higher priority processes.

Memory usage — full operation memory could cause performance degradation.

Disk space — for example when using storage disk as a database, there should be preventive measures to backup the data and free up disk space,

Operating System limits — system's memory, and CPU capabilities.

2.4.4 Error Rate

Error Rate is a metric, which commonly occur in the network systems, especially under high load. During communication between client and server there could be error caused by another network device (router, switch, hub, etc.) or signal disturb. The Error Rate is the mathematical calculation that produces a percentage of problem requests compared to all requests. In the ideal system, there should be zero network error present; however, the reality that there is always be some. Thus usually leads to performance degradation and slow down throughput, because damaged data need to be resend. Error rate is a significant metric because it tells to engineers how many requests failed at a particular point in time of performance testing. This metric is more evident when you can see the percentage of problem strongly increasing, hence you can detect problem easily.

Chapter 3

Messaging Performance Tool

3.1 Measures Process

3.2 Testing Metrics

3.3 Gathered Data and Their Evaluation

3.4 Related Works

SpecJMS

Chapter 4

Analysis and Design

4.1 Qpid-Dispatch Router

4.2 Usable Protocols

AMQP, MQTT - possibly?

4.3 Automatic Topology Generator

4.3.1 Network Components

4.3.2 Structure of Input and Output

4.3.3 Topology Creation

4.4 Qpid-Dispatch Performance Module

4.4.1 [[more subsections about module]]

4.5 Performance and Testing Metrics of Qpid-Dispatch Performance Module

4.6 Gathered Data Evaluation

Chapter 5

Implementation

5.1 Used Technologies

5.1.1 Ansible

5.1.2 Docker

Using for testing Ansible roles (remove?)

5.2 Topology Generator

5.2.1 Template Generator

5.2.2 Generation of Variables

5.2.3 Configuration Files Generation and Deployment

5.3 Qpid-Dispatch Performance Module

5.3.1 TODO - more subsections about implementation

Chapter 6

Experimental Evaluation

6.1 Performance Testing on Various Generated Topology

6.2 Testing results

Chapter 7

Future work and ideas

Chapter 8

Summary

Bibliography

[1] ???: ISTQB Foundation Level and Agile Tester Certification guide. Online. [visited 2017/11/29].
Retrieved from: http://istqbexamcertification.com/

[2] Regression Testing. Online. [visited 2017/11/15].
Retrieved from:
http://softwaretestingfundamentals.com/regression-testing/

[3] Software Testing Dictionary. Online. [visited 2017/11/15].
Retrieved from: https://www.tutorialspoint.com/software_testing_dictionary

[4] Anukool Lakhina, C. D., Mark Crovella: Diagnosing Network-Wide Traffic Anomalies. Online. [visited 2017/11/13].
Retrieved from: http://www.cs.bu.edu/fac/crovella/paper-archive/sigc04-network-wide-anomalies.pdf

[5] Bhatt, N.: Performance Testing – Response vs. Latency vs. Throughput vs. Load vs. Scalability vs. Stress vs. Robustness. Online. [visited 2017/11/5].
Retrieved from: https://nirajrules.wordpress.com/2009/09/17/measuring-performance-response-vs-latency-vs-throughput-vs-load-vs-scalability-vs-stress-vs-robustness/

[6] Broadwell, P. M.: Response Time as a Performability Metric for Online Services. Online. [visited 2017/11/19].
Retrieved from: http://roc.cs.berkeley.edu/papers/csd-04-1324.pdf

[7] Buch, D.: 4 types of load testing and when each should be used. Online. [visited 2017/11/5].
Retrieved from: https://www.radview.com/blog/4-types-of-load-testing-and-when-each-should-be-used

[8] Din, G.: *A Performance Test Design Method and its Implementation Patterns for Multi-Services Systems*. PhD. Thesis. Technical University of Berlin. Berlin, Germany. 2008.

[9] Gao, J.; Ravi, C. S.; Raquel, E.: Measuring Component-Based Systems Using a Systematic Approach and Environment. Online. [visited 2017/10/26].
Retrieved from: https://subs.emis.de/LNI/Proceedings/Proceedings58/GI.Proceedings.58-6.pdf

[10] Kopp, M.: Why Averages Suck and Percentiles are Great. Online. [visited 2017/11/20].

Retrieved from: https://www.dynatrace.com/blog/why-averages-suck-and-percentiles-are-great/

[11] Manzor, S.: Application Performance Testing Basics. Online. [visited 2017/10/26]. Retrieved from: http://www.agileload.com/docs/default-document-library/application-performance-testing-basics-agileload.pdf

[12] Marko Aho, C. V.: Computer System Performance Analysis and Benchmarking. Online. [visited 2017/11/15]. Retrieved from: http://www.cs.inf.ethz.ch/37-235/studentprojects/vinckier_aho.pdf

[13] Martina, K.: *Unified Reporting for Performance Testing*. Master's Thesis. Brno University of Technology, Faculty of Information Technology. Brno. 2017.

[14] Molyneaux, I.: *The Art of Application Performance Testing: Help for Programmers and Quality Assurance*. O'Reilly Media, Inc.. first edition. 2009. ISBN 0596520662, 9780596520663.

[15] Piske, O. R.: Messaging Performance Tool. [Online; visited 2017/10/15]. Retrieved from: http://orpiske.github.io/msg-perf-tool

[16] Red Hat, Inc.. Raleigh, North Carolina, U.S.: *Red Hat JBoss AMQ 7.0 Using AMQ Broker*. 2017.

[17] Red Hat, Inc.. Raleigh, North Carolina, U.S.: *Red Hat JBoss AMQ 7.0 Using AMQ Interconnect*. 2017.

[18] Sharma, D.: Why and How: Performance Test. Online. [visited 2017/10/26]. Retrieved from: http://www.qaiconferences.org/tempQAAC/Why%20&%20How-Performance%20Test.pdf