Institute of Software Technology
Reliable Software Systems Research Group
University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Fachstudie Nr. 188

Performance Testing in Continuous Integration Environments

Chris Geiger Dennis Przytarski Sascha Thullner

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Lars Grunske
Betreuer:	Dipl.-Inform. André van Hoorn
begonnen am:	16.10.2013
beendet am:	17.04.2014
CR-Klassifikation:	D.4.8

Abstract

The purpose of this case study is to evaluate how and which performance testing tools which can be used in continuous integration (CI) environments. By doing so, developers can see the effects of changes immediately and react against performance problems of their applications. This will help companies to eliminate performance issues which the media is reporting about more often every day. CI provides the reference platform for executing the performance tests and the performance testing tools provide metrics like response time and percentage of errors. These metrics can be combined through CI plugins. The results of this combination can be visualized in form of graphs and tables.

Through this case study, we give a short market overview of current CI servers and performance testing tools. In respect of the requirements by adesso AG, we will only evaluate performance testing tools, which can be integrated into the Atlassian Bamboo or Jenkins CI. We evaluated six performance testing tools of which four were integratable into the CI servers. Based on the results of our evaluation we will give a recommendation.

Contents

Chapter 1

Introduction

This chapter provides an overview about the motivation, the goals, the collaboration with our industry partner adesso AG and also the document structure of this case study.

1.1 Motivation

Continuous integration (CI) helps developers to detect functional problems in their code early and therefore allows to increase the quality of the product. Continuous integration is a software development process in which the individual team members integrate their developed code in the whole project on a regular basis [Duvall et al. 2007].

Most of the time, only automated unit tests are executed on CI environments that check whether the functionality of the system is still provided or if some changes in the source code broke parts of a system's functionality. However, this is not the only meaningful information that can be extracted. For example, by creating and executing automated performance tests, it is possible to get even deeper knowledge about a system's dynamic characteristics. Performance Testing in CI environments can help to identify potential performance problems. Developers can get more insight into the performance characteristics of their implementations by running small and automated performance tests. This allows architects, developers, and testers to develop a better feeling on the dynamic behavior of their applications, because they could see the effects of changes immediately. Load tests and roll-outs also get more efficient as a lot of bugs could have already been resolved by analyzing the results from the performance tests running on the CI environment.

1.2 Goals

The goals of this study are the identification of functional and non-functional requirements by adesso for performance testing integrated in CI environments, e.g., concerning support for the specification and automatic execution of performance tests on different levels (unit, integration, system), collection and visualization of results, as well as rules for successful/failing builds. These requirements are represented by the criteria catalog in Chapter 5.

Furthermore, this study investigates the possibility of performance testing in CI environments. This includes the examination of CI servers, monitoring tools, and performance test tools.

The final result should be a scientifically-based recommendation of a combination of a CI environment and a perfomance test tool taylored to adesso's requirements.

1.3 Collaboration with adesso AG

This case study is conducted in collaboration with the adesso AG in Stuttgart. Adesso AG is an independent IT service provider that supports companies through consulting and software development. In cooperation the continuous integration server and performance test tools are selected. Furthermore, a criteria catalog will be created according to the wishes and ideas of the adesso AG.

Adesso AG develops individual software in the programming language Java. By this time, they use Atlassian Bamboo and Jenkins CI as continuous integration servers. Furthermore they are interested in using performance testing tools which can be integrated into their current CI platform. This usage scenario serves us as an environment for this case study.

1.4 Document Structure

The remainder of this document is structured as follows: In Chapter 2, a short introduction is given by describing the foundations of this case study. Chapter 3 describes and arranges the individual phases of the project and the research procedure. Chapter 4 offers a market overview, listing and giving basic information about the different commercial and free CI servers and performance testing tools. Chapter 5 introduces and describes the criteria catalog which will be used to classificate the different performance test tools by their relevance. Chapter 7 gives information about the evaluated tools and explains the results. The last chapter concludes the study and gives a recommendation.

Chapter 2

Foundations

This chapter provides an overview about the foundations about continuous integration and performance testing.

2.1 Continuous Integration

The administration effort of the developement process during a software project grows with its size and complexity. Escpecially the version control and build process are a major challenge. In teams with multiple members, the integration of the various code fragments is an essential task. In this connection often arise so-called integration error. These integration errors are hardly detected during the further development process and thus remain in the software [Duvall et al. 2007].

To avoid these errors and bring other advantages and simplifications, CI is used in software development. CI is a software development process in which the individual team members integrate their developed code in the whole project on a regular basis. Each integration is automatically tested and built, whereby big errors, which may prevent other developers from further development of the software, are detected immediately. This process includes tests which additionally check the functionality of the software [Fowler 2006]. Below the individual components of a CI environment are explained briefly with reference to Figure 2.1.

2.1.1 Version Control System (VCS)

A technical prerequisite for the use of CI is a version control system (VCS). This system manages the source code and other files of all the developers of the software project [Feuste and Schluff 2012]. Popular VCSs are SVN and Git.

2.1.2 Continuous Integration Server

The CI server periodically sends requests to the VCS, weather the current project has been changed. If changes are present, the continuous integration server starts a new build to verify the new code for errors. A build consists of compiling the source code and executions of the associated tests, such as unit tests [Feuste and Schluff 2012].

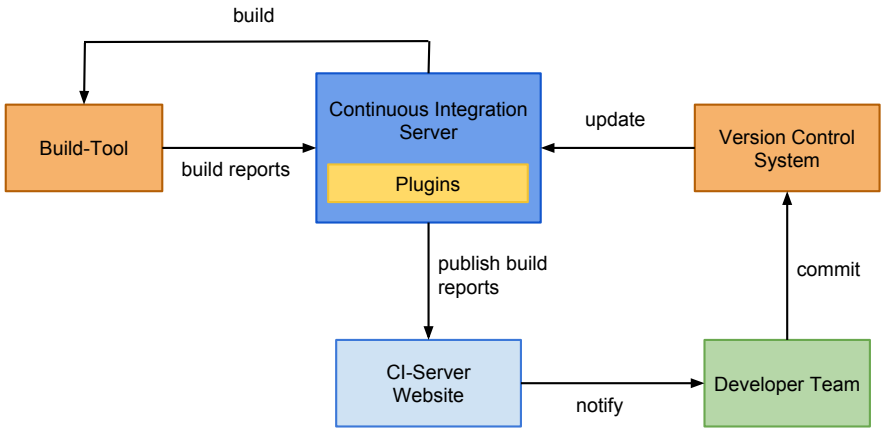


Figure 2.1. Continuous Integration Components [Feuste and Schluff 2012]

2.1.3 Build Tool

The CI server performs the compilation and the test execution not by itself. This functionality is offered by a build tool. This tool must be set up by the development team prior to using the CI server. The necessary files for using the build tool are often managed in the VCS as well [Feuste and Schluff 2012]. Popular build tools are Apache Ant and Maven.

2.1.4 Plugins

For the continuous integration server, there are a variety of plugins that can extend its functionality. For example the continuous integration server Jenkins has over 600 plugins available like a Git, Findbugs or, Checkstyle plugin.

2.1.5 CI-Server Website

The results of the build process with associated data and information are stored on the continuous integration server as reports. These reports are available via a website. They provide, for example, detailed information about errors, test results and code quality [Feuste and Schluff 2012].

2.2 Performance Testing

Performance testing is part of software engineering which continuously increases in importance. Performance testing determines how a system behaves under a certain

workload. There are several types of performance testing performed for various reasons, which will be described in the following based on Meier et al. 2007.

2.2.1 Load Testing

Load testing examines the behavior of a system under a specific expected normal load. An example of a load test is the simultaneous access of several users to a website, which then perform specific actions. Results of this test are, for example, the server response time or identified bottlenecks.

2.2.2 Stress Testing

Stress testing is useful to find the robustness and reliability of a system. It determines or validates the behavior of an application beyond normal or peak load conditions.

2.2.3 Capacity Testing

A capacity test provides information about how workload can be handled and determines capacity limits.

2.2.4 Component Testing

Every test which targets an architectural component of an application is called component test. For example servers, databases or storage devices.

2.2.5 Unit Testing

Unit tests during performance testing hardly differ from normal unit tests. They are only optimized for performance aspects.

Chapter 3

Research Procedure

This chapter describes the research procedure, which was conducted over six months in the period between October, 16th 2013 and April, 17th 2014. Regular meetings to discuss the recent results and next steps with our adviser took place bi-weekly. Our case study was subdivided into four milestones. We performed this study in a team of three members. Figure 3.1 visualizes our schedule as a Gantt chart.

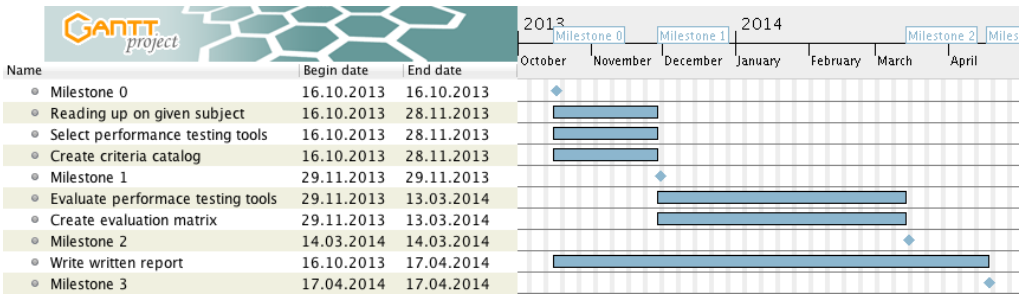


Figure 3.1. Timeline as Gantt Diagram

3.1 Kick-Off Meeting

Because of a short term appointment with our industry partner, adesso AG, we had just a small preparation phase before the kick-off milestone. Our goal for the first milestone was to present the motiviation and a small market overview of some performance testing in continuous integration enviroments. Our deadline for the 0th milestone was on October, 16th 2013. At this point, it was decided that we are only looking for performance testing tools which are usable with Atlassian Bamboo or Jenkins CI.

3.2 Research and Criteria Catalog

After the kick-off meeting, we had a phase for working into our case study topic, creating a market overview and the criteria catalog. During this phase, we found many interesting

performance testing tools which we have categorized either as for functional or web testing. Our deadline for the 1st milestone was on 29th November 2013.

3.3 Evaluation of the Performance Testing Tools

To establish well-founded results, we installed each listed performance testing tool on its corresponding continuous integration platform. We noted for each performance tool how each satisfied the criteria catalog. Our deadline for the 2nd milestone was on March, 14th 2014.

3.4 Final Presentation and Written Report

After we had finished with our evaluation phase, we held a final presentation to present our results at adesso AG in Stuttgart. We also wrote this written report to sum up our evaluation. Our deadline for the 3rd milestone was on April, 17th 2014.

Chapter 4

Market Overview

This chapter provides an overview of the tools for this case study which we found by an internet research. These tools are divided into two sections — one for the continuous integration servers (Section 4.1) and one for the performance testing tools (Section 4.2). Each of these sections includes a table with the attributes of the tool and a short description which was taken from the corresponding website. The unit testing tools as shown in Table 6.1 are limited to the programming language Java. The table includes the following attributes:

- Name
  The name of the software.

- Developer
  The person or organization who is maintaining the software.

- Operating System
  The system on which the software can be run.

- License
  The license under which the software can be used.

- Website
  The URL of the website.

4.1 Continuous Integration

This section provides an overview of known and mostly used continuous integration servers in alphabetic order.

4.1.1 Bamboo

Name	Bamboo
Developer	Atlassian
Operating System	Cross-platform
License	Proprietary
Website	atlassian.com/software/bamboo

Bamboo is a product of a whole range of Atlassian products. It is a continuous integration server which can be connected with the other products by Atlassian. In combination with these products it covers all issues of an underlying software project.

4.1.2 Continuum

Name	Apache Continuum
Developer	Apache Software Foundation
Operating System	Cross-platform
License	Apache License 2.0
Website	continuum.apache.org

Apache Continuum is a continuous integration server that provides a number of functions. Among other things, automated builds, a management for releases and role based safety functions.

4.1.3 CruiseControl

Name	CruiseControl
Developer	ThoughtWorks
Operating System	Cross-platform
License	BSD 3-Clause License
Website	cruisecontrol.sourceforge.net

On the one hand CruiseControl is a continuous integration server on the other hand it is a framework which is extensible to create a custom continuous build

process. There are many plugins to extend CruiseControl in areas such as source control or notification management.

4.1.4 Hudson

Name	Hudson
Developer	Eclipse Foundation
Operating System	Cross-platform
License	Eclipse Public License
Website	hudson-ci.org

Hudson is a continuous integration server. It mainly focuses on building and testing software projects continuously. It can also monitor the execution of jobs which were run externally.

4.1.5 Jenkins

Name	Jenkins (Hudson Fork)
Developer	Community
Operating System	Cross-platform
License	MIT License
Website	jenkins-ci.org

Jenkins is a fork of hudson. It is also a continuous integration server. Since it is a fork of hudson these two continuous integration servers are quite similar.

4.1.6 TeamCity

Name	TeamCity
Developer	JetBrains
Operating System	server-based web application
License	Proprietary
Website	jetbrains.com/teamcity

TeamCity provides a REST API and over 100 plugins to extend its basic functionality.

4.2 Performance Testing Tools

This section provides an overview of known and mostly used performance testing tools for continuous integration servers in alphabetical order.

4.2.1 Blitz

Name	Blitz
Developer	Spirent
Operating System	Cross-platform
License	Proprietary
Website	blitz.io

Blitz is a load testing tool for websites, web applications, and REST APIs.

4.2.2 ContiPerf 2

Name	ContiPerf 2
Developer	Volker Bergmann
Operating System	Cross-platform
License	BSD License (and 3 others)
Website	databene.org/contiperf.html

ContiPerf 2 is a performance testing tool that makes it possible to extend JUnit 4 test cases as performance tests.

4.2.3 JMeter

Name	Apache JMeter
Developer	Apache Software Foundation
Operating System	Cross-platform
License	Apache License 2.0
Website	jmeter.apache.org

Apache JMeter can be used for performance tests like simulating a heavy load on a server or to test performance on static and dynamic resources.

4.2.4 JUnitPerf

Name	JUnitPerf
Developer	Mike Clark (clarkware)
Operating System	Cross-platform
License	BSD 3-Clause License
Website	clarkware.com/software/JUnitPerf.html

JUnitPerf is a tool to extend existing JUnit tests to measure the performance and scalability of the functionality.

4.2.5 ScalaMeter

Name	ScalaMeter
Developer	Aleksandar Prokopec, Josh Suereth
Operating System	Cross-platform
License	BSD 3-Clause License
Website	axel22.github.io/scalameter

ScalaMeter is a framework for performance regression testing. It can be used for Scala and Java.

4.2.6 Selenium

Name	Selenium
Developer	ThoughtWorks
Operating System	Cross-platform
License	Apache License 2.0
Website	docs.seleniumhq.org

Selenium is a tool to automate interactions with web applications. With this automation it is also possible to test the performance of web applications.

Chapter 5

Criteria Catalog

This chapter provides the criteria catalog for the rating of the tools. Because the continuous integration servers are already given, namely Bamboo and Jenkins, this criteria catalog will be only applied on the performance testing tools.

5.1 Knock-out Criteria

The knock-out criteria are of nominal type. There is only one simple yes or no criteria which must be satisfied.

- Integration into continuous integration server Atlassian Bamboo or Jenkins CI must be possible.

5.2 Relevant Criteria

The relevant criteria are of ordinal type. Each criteria can get a positive ($\oplus$), neutral ($\odot$) or negative ($\ominus$) rating. A criteria with a positive rating gets two points, a neutral rating gets one point and a negative rating gets zero points.

5.2.1 Documentation

If a good documentation of the product exists, it is likely that it is widely used.

$\oplus$	The product provides a well-written documentation.
$\odot$	The product provides abbreviated documentation.
$\ominus$	The product provides absurd or even no documentation.

5.2.2 Community

If a community of the product exists, it is likely that it is widely distributed. Is it possible to get answers to specific questions from any community?

⊕	People blog about the product and ask questions on forums and websites like Stack Overflow.
⊙	Product owner administrates a help site where questions may be answered. It is tempted to establish a community.
⊖	No community established as yet and product owner does not care.

5.2.3 Integration into the Continuous Integration Platform

Is the performance testing tool easily integratable into Atlassian Bamboo or Jenkins CI?

⊕	Seamless integration by using the provided documentation without hassle.
⊙	Integration was after some made inquiries successful.
⊖	Integration looks like an impossible mission.

5.2.4 Learnability

How easy is it to learn the product?

⊕	Efficient and error-free interaction with the product was relative quick possible.
⊙	Not that easy to learn, so that it could be improved.
⊖	Product is not easy to learn.

5.2.5 Likeability

Is the product likeable?

⊕	It is fun to work with the product.
⊙	The produt is okay to use but nothing special.
⊖	The product is annoying and not worth to use it.

5.2.6 Maintenance

Is the product maintained by the origin product owner?

⊕	A recent stable version exists and the next version is under active development.
⊙	A usable version exists and bugfix support is provided.
⊖	A usable version without any further bugfix support exists.

5.2.7 Software Cost

Is the tool free or affordable?

⊕	Any open source license compatible with the Bamboo and/or Jenkins license.
⊙	Product price is announced publicly.
⊖	Product price is only available on request.

5.2.8 Support

Is it possible to receive at least basic support?

⊕	Product owner provides basic free support like a forum, mailing list, or wiki.
⊙	Only paid support is available.
⊖	No support is offered.

5.2.9 Usability

Can the user complete key tasks with no unanswered questions?

⊕	The most general principles for interaction design are adhered to.
⊙	Some principles for interaction design are adhered to.
⊖	No principles for interaction design are adhered to.

Chapter 6

Evaluation

This chapter is divided into four sections. The first section provides a evaluation of each performance testing tool. The second section investigates the expandability of the CI servers. The third section investigates the integration of the performance testing tools into the CI servers. The last section summarizes the results of the previous section.

6.1 Performance Testing Tools

In Table 6.1 we classify the integratable performance testing tools in their corresponding testing types (rows) and levels (columns).

Table 6.1. Classification of the performance testing tools without ScalaMeter and Selenium

	Unit	**Integration**	**System**
Capacity, Load, Stress	ContiPerf 2, JUnitPerf	-	Blitz, JMeter

In this section, we evaluate the performance testing tools from the market overview in Chapter 4. Figure 6.1 shows a Java method that factorizes a given number into prime numbers. We use this code to run performance tests against this method, if it is possible with the performance testing tool.

```java
public class PrimeFactorization {
   public static List<Long> getPrimeFactors (long number) {
      List<Long> primeFactors = new ArrayList<Long>();
      for (long i = 2; i <= number; i++) {
         while (0 == number % i) {
            primeFactors.add(i);
            number /= i;
         }
      }
      return primeFactors;
   }
}
```

Figure 6.1. Prime factorization in Java [*Prime Factorization - Algorithm in Java*]

6.1.1 Blitz

Blitz is a cloud-based load and performance testing service. It is used to test a web API or web app with simulated visitors from around the world. Blitz is able to perform sprints, which are a simple HTTP or SSL requests to a specific URL and rushes, which are multiple sprints within a specific pattern. Figure 6.2 shows a rush with a pattern where a different amount of users perform a request over a specific time.

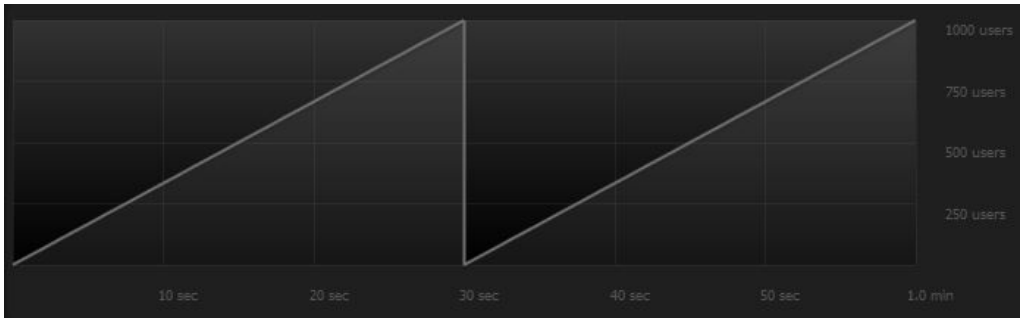


Figure 6.2. Example of a rush in form of a saw tooth pattern

To avoid accidental denial-of-service attacks on different websites Blitz users have to authorize their application or website on the Blitz website and place a unique file in their web server root. To perform a sprint or a rush the Blitz bar shown in Figure 6.3 can be used. There are different parameters for the request which can be e.g., URL, method, region, users or duration.

By pressing the play button the request will be performed and Blitz returns detailed

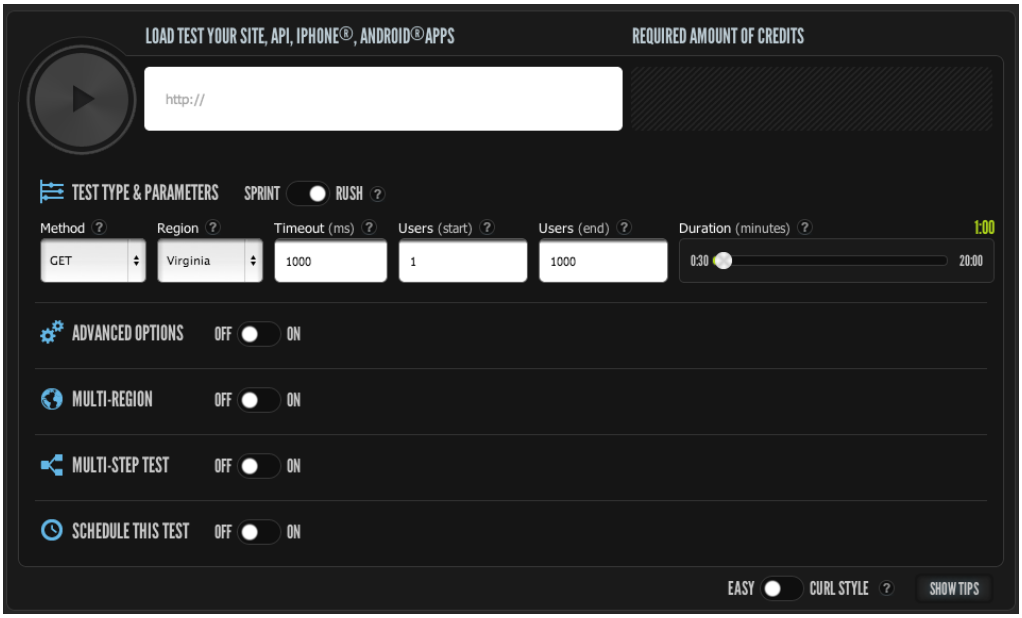


Figure 6.3. Blitz-Bar

information about its execution. For example Blitz provides information about the application performance with metrics such as response time, hit rates, errors, timeouts and also gives clues to help scale out an application. A full output of Blitz is shown in the appendix in Figure 1 to Figure 3. The view of information can be enhanced by using plugins. For Blitz there are several plugins like New Relic, Scout, and CopperEgg each giving different additional information like CPU utilization or troughput. Blitz is very useful to test applications in production mode.

Furthermore Blitz provides clients for many different platforms like Java, Maven, JavaScript, and PHP. After integrating the client it is possible to execute sprints and rushes directly within the code. Because it is a cloud based service, the load is generated by the server network of Blitz. Figure 6.4 shows how simple it is to perform a rush to the website www.test.de with users from California in form of the saw tooth pattern explained above.

6.1.2 ContiPerf 2

ContiPerf 2 is tool to transform JUnit tests into performance tests. To enable ContiPerf2 for a JUnit test case we simply add a static @Rule at the beginning as seen in Figure 6.5. The second step is to add the execution parameters (@PerfTest) and performance requirements (@Required). In the example the test is configured to be executed 2000 times with 20

```
public class BlitzApp {
      public static void main( String[] args ) {
            try {
                  Rush r1 = (Rush) io.blitz.command.Curl.parse(
                        USERNAME,
                        API_KEY,
                        "-p 1-1000:30,1-1000:30 -r california http://www.test.de"
                        );
                  r1.addListener(new IRushListener() {
                        public boolean onStatus(RushResult result) {
                              System.err.print(".");
                        }
                        public void onComplete(RushResult result) {
                              System.err.println("SUCCESS!");
                        }
                  });
                  r1.execute();
            } catch (Exception ex) {
                  System.err.println("Rush failed.");
            }
      }
}
```

Figure 6.4. Performing a rush over the java client

concurrent threads. Each thread does 100 invocations. The execution time must be within 7000 milliseconds (7 seconds) and at least 300 test executions per second are required.

```
public class ContiPerfTest {
   @Rule
   public ContiPerfRule i = new ContiPerfRule();

   @Test
   @PerfTest(invocations = 2000, threads = 20)
   @Required(throughput = 300, totalTime = 7000)
   public void PrimeFactorTest() {
      PrimeFactorization.getPrimeFactors(Long.MAX_VALUE);
   }
}
```

Figure 6.5. First prime factorization test with ContiPerf2 in Java

The output shows that 2000 invocations were executed, that the maximum execution

time of an invocation was 210 ms, the average execution time of the invocations was 71.35 ms and the median was 60.

```
test.ContiPerfTest.PrimeFactorTest
samples: 2000
max:   210
average: 71.3475
median: 60
```

Figure 6.6. First test output

The test failed due to a timeout. The condition of a maximum execution time of 7000 milliseconds was not met.

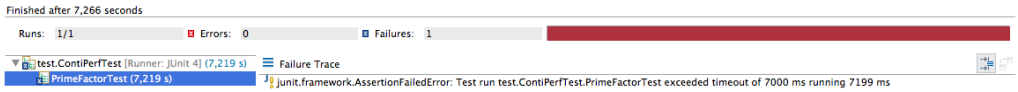


Figure 6.7. First test result

We executed this test three times with different conditions.
The second time we changed the performance requirements from a maximum execution time of 7000 milliseconds to 8000 milliseconds.

```
public class ContiPerfTest {
   @Rule
   public ContiPerfRule i = new ContiPerfRule();

   @Test
   @PerfTest(invocations = 2000, threads = 20)
   @Required(throughput = 300, totalTime = 8000)
   public void PrimeFactorTest() {
      PrimeFactorization.getPrimeFactors(Long.MAX_VALUE);
   }
}
```

Figure 6.8. Second prime factorization test with ContiPerf2 in Java

The output of the first and the second test was nearly the same.

```
test.ContiPerfTest.PrimeFactorTest
samples: 2000
max:   237
average: 73.821
median: 63
```

Figure 6.9. Second test output

This time the maximum execution time was within the required parameters but the test had only 268 calls per second, 300 calls were required.

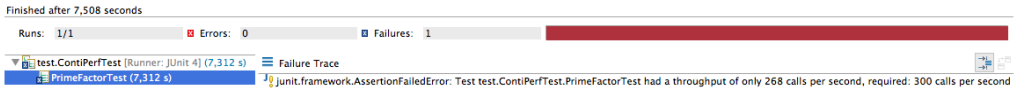


Figure 6.10. Second test result

We changed the required parameters again.

```
public class ContiPerfTest {
   @Rule
   public ContiPerfRule i = new ContiPerfRule();

   @Test
   @PerfTest(invocations = 2000, threads = 20)
   @Required(throughput = 200, totalTime = 8000)
   public void PrimeFactorTest() {
      PrimeFactorization.getPrimeFactors(Long.MAX_VALUE);
   }
}
```

Figure 6.11. Third prime factorization test with ContiPerf2 in Java

```
test.ContiPerfTest.PrimeFactorTest
samples: 2000
max:   288
average: 72.1655
median: 60
```

Figure 6.12. Third test output

The third execution of our test met all the required conditions and we got a positive result.

Figure 6.13. Third test result

6.1.3 JMeter

JMeter is a Java desktop application which is used to simulate a heavy load on both static and dynamic resources like databases, FTP servers, web dynamic languages [*Apache JMeter*]. It is highly configurable but the user interface (Figure 6.14) takes much getting used to. Because of that, the user of this tool needs a long training period for a proper use to set up a test plan.

A test plan executes a series of steps. If we want to test a web site, a test plan as an example may look like this:

First, we create a thread group, which contains the number of threads, ramp-up period and loop count properties. The first property specifies, how many threads will execute the test plan completely independently of other threads. Each thread can be seen as a visitor of the web site. The second specifies, how long it will take to reach the given number of threads. This ensures, that the load can increase safely. The last property specifies, how often a thread executes the test plan. After that, we need to create a HTTP request containing the server address, port, and path. To see some results, we add a listener, which collects the results and produces a graph. Finally, we can load test a web site with this configuration. Figure 6.15 shows an example of a graphical result of this test plan.

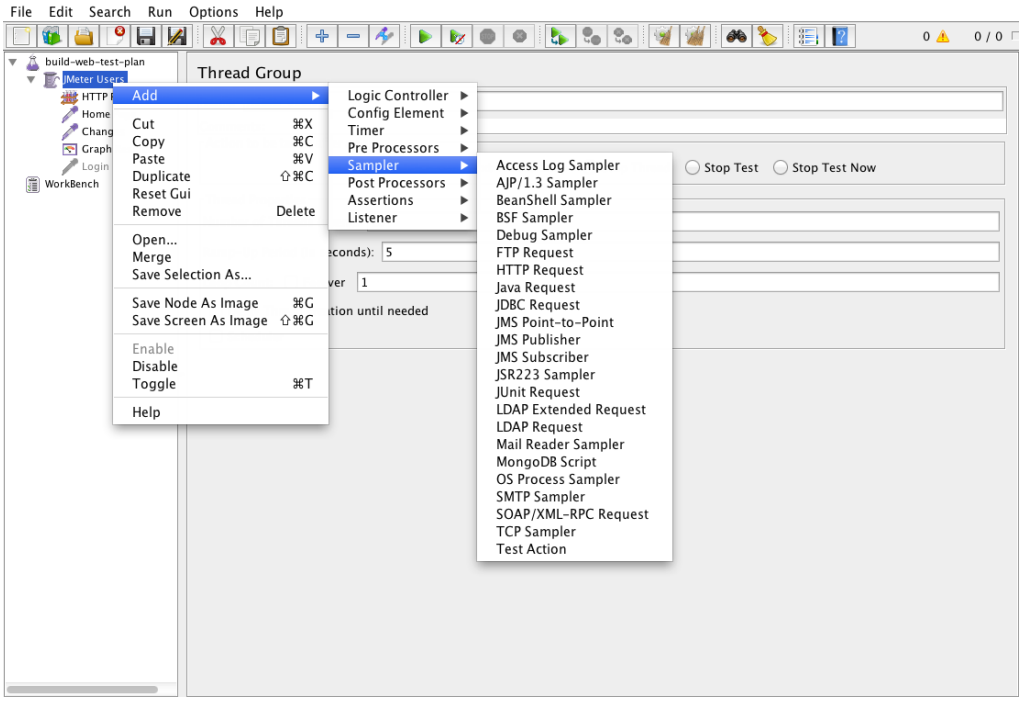


Figure 6.14. JMeter graphical user interface

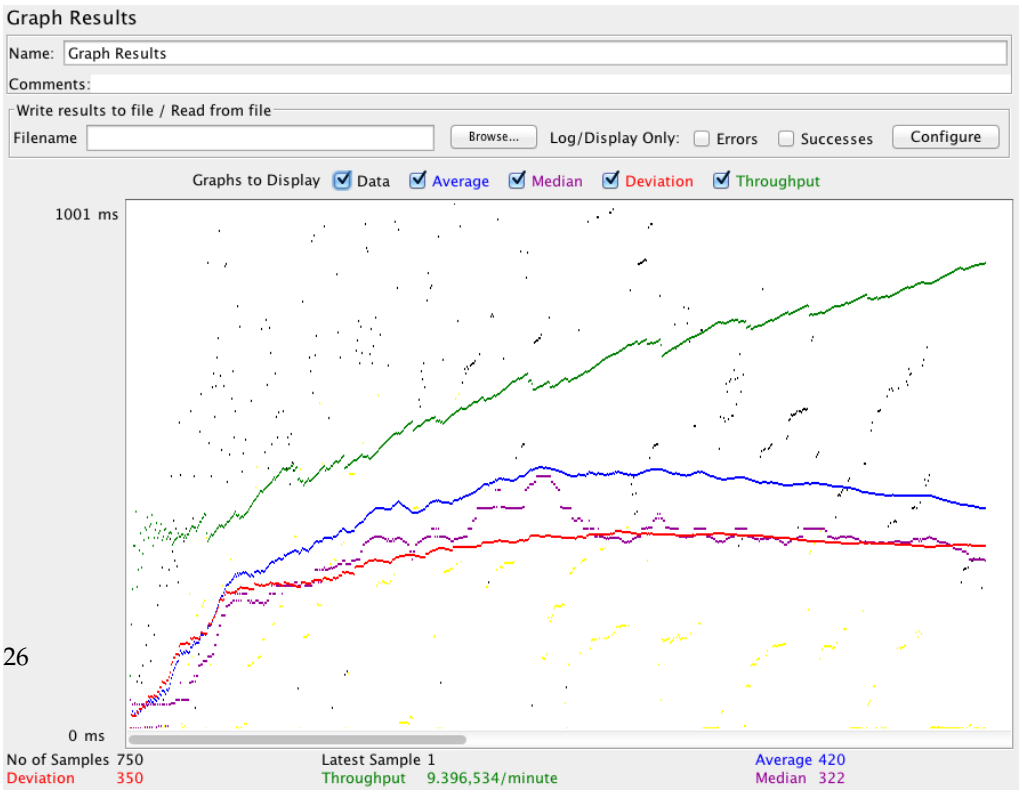


Figure 6.15. Graphical results of a test plan for the phpMyAdmin main screen web site with following properties: number of threads: 125, ramp-up period: 2, loop count: 3.

6.1.4 JUnitPerf

JUnitPerf is a collection of JUnit test decorators. It can be used to measure the performance and the scalability of functionality contained within JUnit tests [*JUnitPerf*]. The installation of JUnitPerf is very simple. We just had to put the .jar into the Eclipse build path and we were ready to start testing. We decided to implement two simple load tests. Running a test under a specific load in JUnitPerf means running a test with a simulated number of cuncurrent users and iterations. In each test we simulated 2500 cuncurrent users. The first test measures the troughput of the function in Figure 6.1 under a specific load which is shown in Figure 6.16.

```
public class PrimeFactorThroughputUnderLoadTest {
   public static Test suite() {
      Test test = new PrimeFactorTest("primeFactors");
      Test loadTest = new LoadTest(test, 2500);
      Test timedTest = new TimedTest(loadTest, 8000);
      return timedTest;
   }
   public static void main(String args[]) {
      junit.textui.TestRunner.run(suite());
   }
}
```

Figure 6.16. Testing troughput under load with JUnitPerf

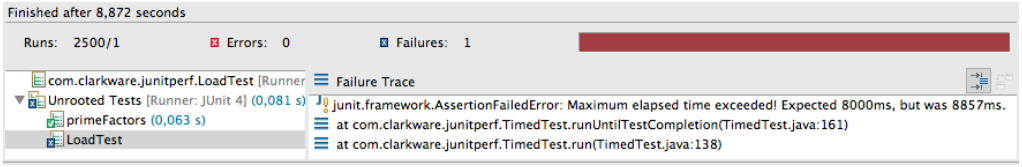


Figure 6.17. Result of the troughput under load test

The second test measures the response time under a specific load of the same function which is shown in Figure 6.18.

```
public class PrimeFactorResponseTimeUnderLoadTest {
      public static Test suite() {
            Test test = new PrimeFactorTest("primeFactors");
            Test timedTest = new TimedTest(test, 20);
            Test loadTest = new LoadTest(timedTest, 2500);
            return loadTest;
      }
      public static void main(String args[]) {
            junit.textui.TestRunner.run(suite());
      }
}
```

Figure 6.18. Testing response time under load with JUnitPerf

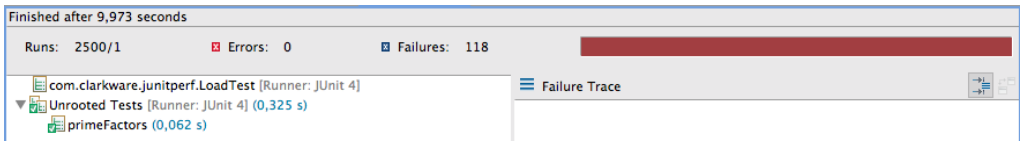


Figure 6.19. Result of the response time under load test

6.1.5 ScalaMeter

ScalaMeter is a microbenchmarking and performance regression testing framework which can be used for Java and Scala. It can measure the running time of a method or algorithm that is run against some input. [*ScalaMeter, Automate your performance testing today.*] Unfortunately, the tests in ScalaMeter must be written in Scala. We tried to write a test in Java but without success. Figure 6.20 shows such a test in Scala. The test will run ten rounds to achieve usable benchmarking results. The first round will test the function in Figure 6.1 with numbers from range 1 to 1000. With each additional round the range will be incremented by 1000. For every round the elapsed time will be measured. After the last round the benchmark results are outputted to the console as seen in Figure 6.21.

```
object RangeBenchmark extends PerformanceTest.Quickbenchmark {
 val numbers = Gen.range("number")(1000, 10000, 1000)
 val ranges = for {
  number <- numbers
 } yield 1 until number
 performance of "PrimeFactorization" in {
  measure method "getPrimeFactors" in {
   using(ranges) in {
    r => r.foreach(PrimeFactorization.getPrimeFactors(_))
   }
  }
 }
}
```

Figure 6.20. ScalaMeter RangeBenchmark in Scala [*ScalaMeter, Automate your performance testing today.*]

```
::Benchmark PrimeFactorization.getPrimeFactors::
Parameters(number -> 1000): 0.503
Parameters(number -> 2000): 1.74
Parameters(number -> 3000): 3.592
Parameters(number -> 4000): 6.019
Parameters(number -> 5000): 9.271
Parameters(number -> 6000): 12.414
Parameters(number -> 7000): 16.588
Parameters(number -> 8000): 21.11
Parameters(number -> 9000): 26.296
Parameters(number -> 10000): 32.267
```

Figure 6.21. Test results of benchmarking the function getPrimeFactors in milliseconds

6.1.6 Selenium

Selenium is in the first place a tool to automate a browser. We tried it with Firefox and started to test some performance issues. After a few minutes it was obvious that this tool is not a typical performance testing tool. Our test can be seen in the code below.

```
public class SeleniumTest extends SeleneseTestCase {
   public void setUp() throws Exception {
      setUp("https://www.google.de/", "*firefox");
   }

   public void testNew() throws Exception {
        selenium.open("/");
        selenium.type("q", "selenium rc");
        selenium.click("btnG");
        selenium.waitForPageToLoad("30000");
        assertTrue(selenium.isTextPresent("Selenium"));
   }
}
```

Figure 6.22. First test with Selenium in Java

In this test we tried to open Google in Firefox and executed a search request for "selenium rc". If the response page would contain the word "Selenium" the test would be passed if the timout of 30 seconds would not be exceeded. Google is importing its results dynamically with jQuery and because of that Selenium did not pass this test although it found some results for "selenium rc". Selenium is as said before more likely a tool to automate the browser to perform repeating task automatically. We stopped to test selenium any further.

6.2 Expandability of Bamboo and Jenkins

6.2.1 Atlassian Bamboo

The extensions in Atlassian Bamboo are called add-ons. A marketplace[1] exists where these add-ons can be found. It is also possible to install these add-ons directly from the web interface (Figure 6.23).

Figure 6.23. Marketplace in Atlassian Bamboo

[1] https://marketplace.atlassian.com/plugins/app/bamboo

By searching the marketplace for the term performance, we found an add-on called blitz.io which accords with a performance testing tool by name.

6.2.2 Jenkins CI

The extensions in Jenkins CI are called plugins. There are about 1.000 plugins which all can be found on the plugins website[2]. It is also possible to install these plugins directly from the web interface (Figure 6.24).

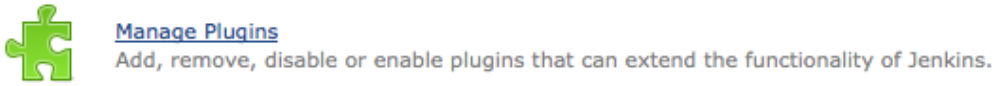


Figure 6.24. Manage Plugins in Jenkins CI

By going through the plugins list, we found a plugin called Performance Plugin, which captures reports from the JMeter XML or JUnit format and generates graphical charts with a trend report of performance (Figure 6.25). [*Jenkins Performance Plugin*]

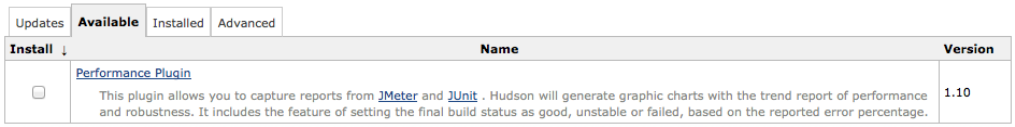


Figure 6.25. Available Plugins in Jenkins CI

6.3 Integration of Performance Testing Tools in Bamboo and Jenkins

In this section, we try to integrate the performance testing tools into the CI server which is either Atlassian Bamboo or Jenkins CI.

After some investigation, we found out that some testing tools are not integratable in neither Bamboo nor Jenkins. Table 6.2 shows the result of our investigation.

[2] https://wiki.jenkins-ci.org/display/JENKINS/Plugins

Table 6.2. Integration of the performance testing tools

Performance Testing Tool	CI Server
Blitz	Atlassian Bamboo, Jenkins CI
ContiPerf 2	Jenkins CI
JMeter	Atlassian Bamboo, Jenkins CI
JUnitPerf	Jenkins CI
ScalaMeter	-
Selenium	-

6.3.1 Plugin Overview

This section provides an overview of the plugins for this case study which we evaluated. Each of these tools is described in a table with the attributes of the tool. The table includes the following attributes:

- Name
  The name of the plugin.

- Developer
  The person or organization who is maintaining the plugin.

- CI-Server
  The CI-Server on which the plugin can be run.

- License
  The license under which the plugin can be used.

- Website
  The URL of the website.

Name	blitz.io
Developer	Mu Dynamics
CI-Server	Bamboo
License	Commercial - no charge
Website	marketplace.atlassian.com/plugins/io.blitz.bamboo-plugin

Name	JMeter Aggregator for Bamboo
Developer	Atlassian Labs
CI-Server	Bamboo
License	BSD License
Website	marketplace.atlassian.com/plugins/jmeterAggregator

Name	Blitz_io
Developer	Manuel Carrasco Monino
CI-Server	Jenkins
License	Unknown
Website	wiki.jenkins-ci.org/display/JENKINS/Blitz_io

Name	Performance Plugin
Developer	Mu Dynamics
CI-Server	Jenkins
License	Unknown
Website	wiki.jenkins-ci.org/display/JENKINS/Performance+Plugin

6.3.2 Blitz

To use the Blitz add-on for Atlassian Bamboo or the plugin for Jenkins CI a registration at blitz.io is needed. Blitz assigns a API username and API key to each user. These credentials are needed to perform a sprint or a rush through the add-on or plugin. Furthermore the application which should be tested has to be authorized as described in Section 6.1.1.

Atlassian Bamboo

During our research we found out that the current version of the Blitz add-on is not compatible with the newest Atlassian Bamboo 5.4.1. We contacted the developers of Blitz, which informed us that their add-on will be adapted in February 2014. In March we decided to evaluate the Blitz add-on with Atlassian Bamboo 3.3.4 because this was the last compatible version. It is very simple to integrate Blitz into Atlassian Bamboo. After searching the Atlassian marketplace for the term "Blitz", the add-on can be installed with one click on "Install Now" as described in section 6.2.1. A sprint or a rush can be added to a job in Atlassian Bamboo by creating a new task and choose "Blitz Curl" as shown in Figure 6.26.

There the "Blitz Curl Configurations" can be set. For each "Blitz Curl", the command, the API username, the API key and the value of maximum errors in percent have to be specified. Figure 6.27 shows a possible configuration for a rush.

After the execution of the rush, Atlassian Bamboo shows two diagrams with information about the response time and the hit rate as show in Figure 6.28. The response time diagram is showing the response time (light green line) of the server during one minute with an increasing amount of concurrent users (grey line). The hit rate diagram shows the successfull hits (green line), the timeouts (orange line) and the errors (red line) during one minute with an increasing amount of concurrent users (grey line).

Jenkins CI

Integrating Blitz into Jenkins is as easy as integrating Blitz into Atlassian Bamboo. After searching the plugin manager in Jenkins for the term "Blitz", the plugin can be installed be checking its checkbox and clicking "install without restart" as described in section 6.2.2. To use the Blitz plugin for Jenkins CI the credentials from blitz.io must be set under "Configure System" within the Jenkins CI settings as shown in Figure 6.29.

The actual sprint or rush can be configured within the settings of the Jenkins CI project. To add a rush or a sprint to the project, the option "Blitz.io" under the category "Post-build Actions" has to be chosen. Figure 6.30 shows a possible configuration of a rush within Jenkins CI.

After the execution of the rush, Jenkins CI gives additional textual information in opposite to Atlassian Bamboo. These information contain a short summary about timeouts, errors and successfull hits and escpecially a pie diagram as shown in Figure 6.31

Furthermore Jenkins shows the same two diagrams as Atlassian Bamboo with information about the response time and the hit rate as show in Figure 6.32. The response time diagram is showing the response time (light green line) of the server during one minute with an increasing amount of concurrent users (grey line). The hit rate diagram show the successfull hits (green line), the timeouts (orange line) and the errors (red line) during one minute with an increasing amount of concurrent users (grey line).

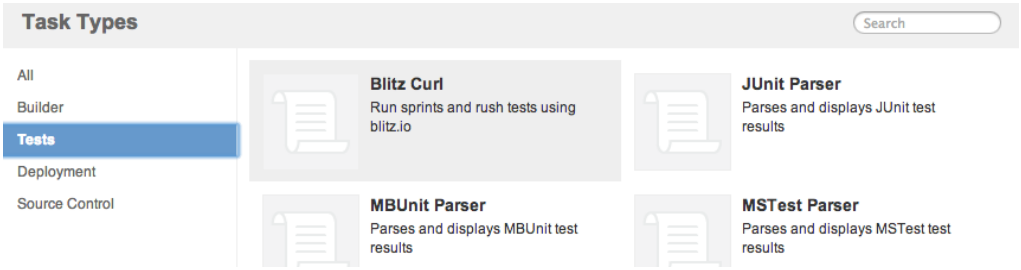


Figure 6.26. Adding a new task to a job in Atlassian Bamboo

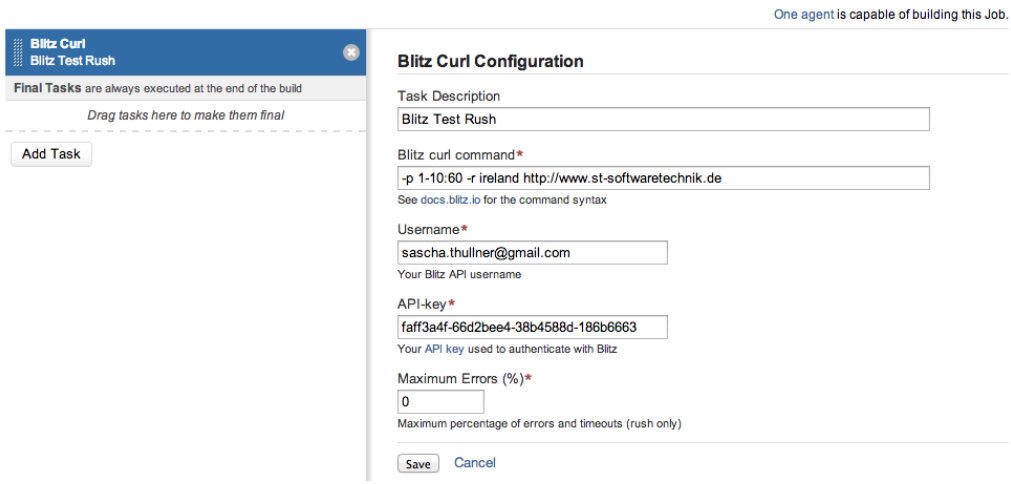


Figure 6.27. Possible configuratuion for a rush in Atlassian Bamboo

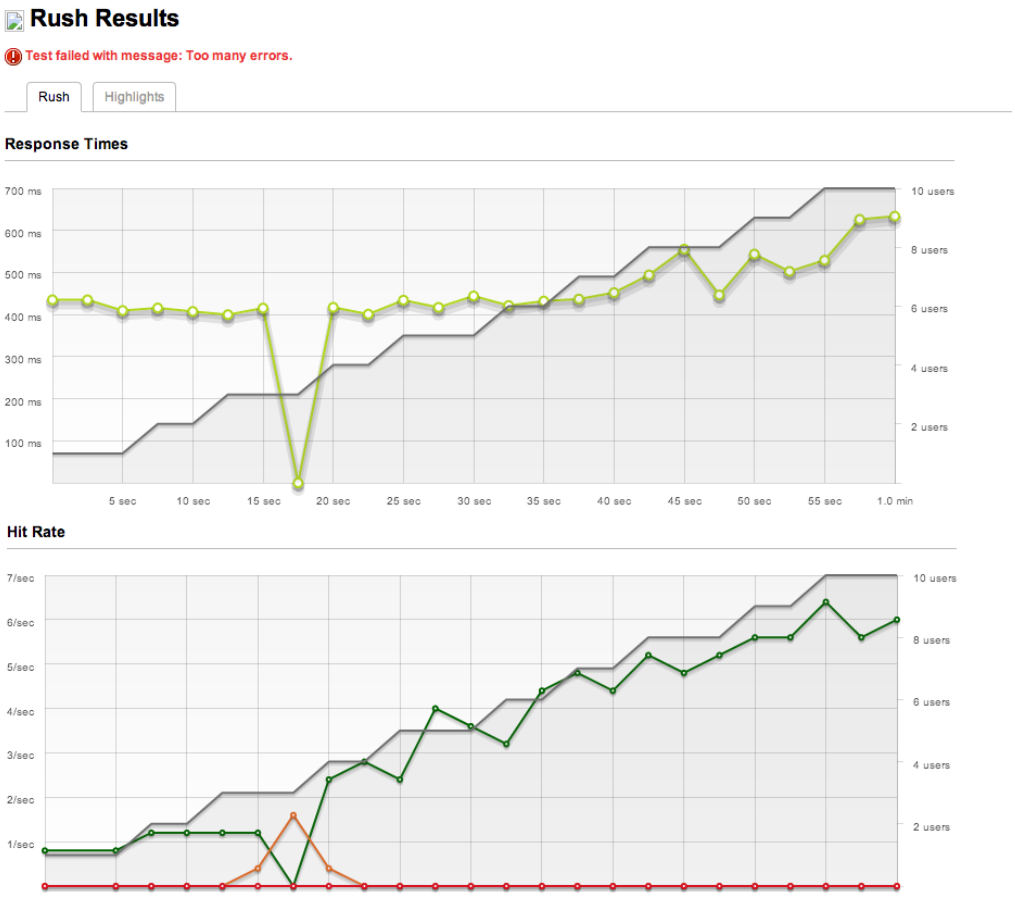


Figure 6.28. Result of a rush execution in Atlassian Bamboo

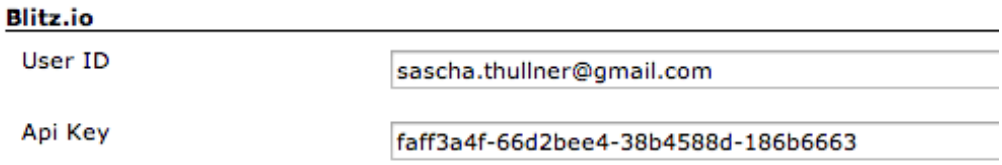


Figure 6.29. API username and API key in the Jenkins CI settings

Figure 6.30. Configure a rush within Jenkins CI

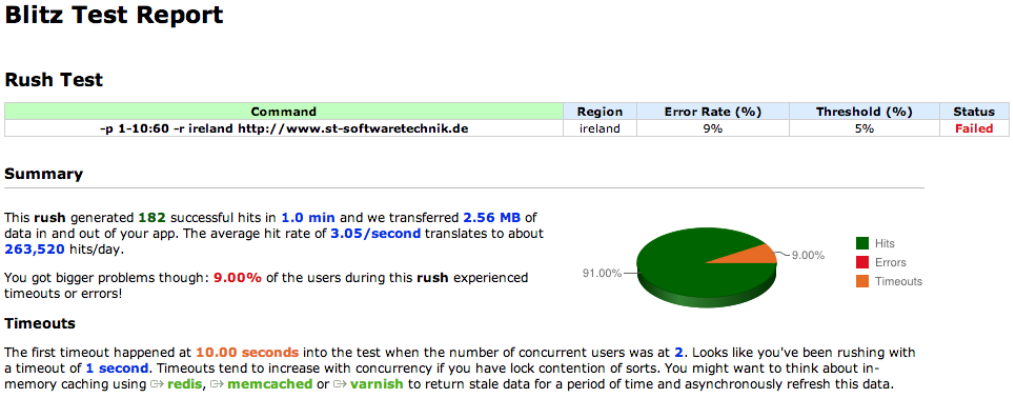


Figure 6.31. Result of a rush execution in Jenkins - part 1

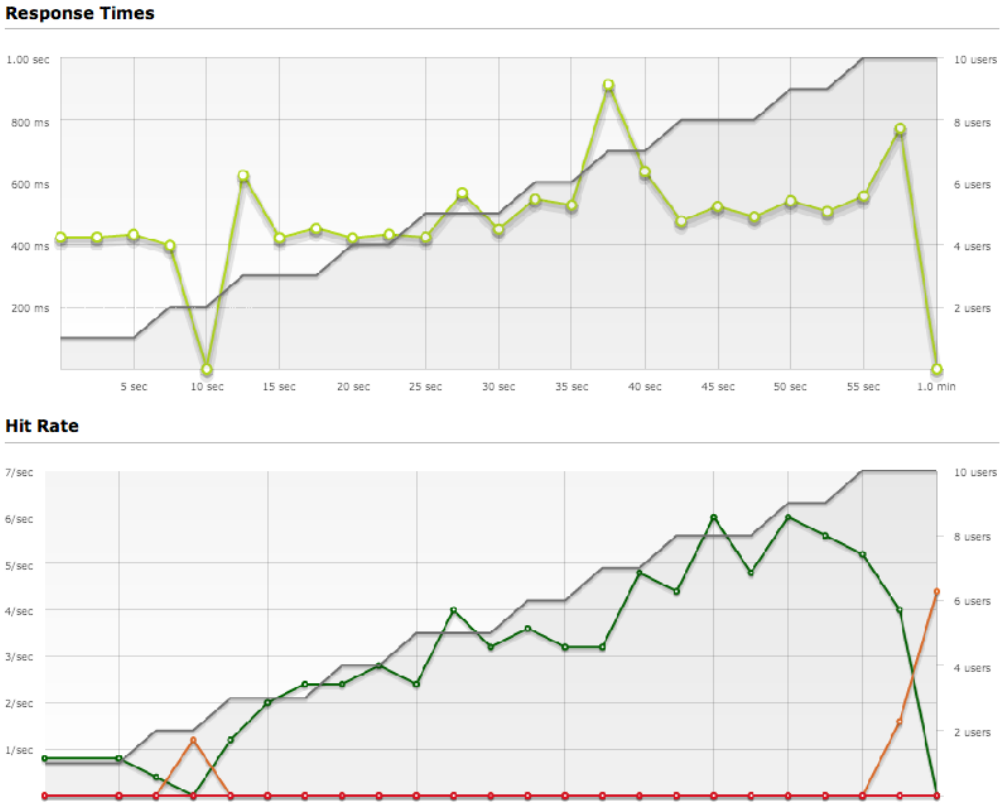


Figure 6.32. Result of a rush execution in Jenkins - part 2

6.3.3 ContiPerf 2 and JUnitPerf

We decided to describe ContiPerf2 and JUnitPerf in only one section because the usage and the integration are quite similar. The integration of these tools was only possible in Jenkins. We were not able to use them with Bamboo because there are no add-ons available.

To integrate one of these two performance tools into Jenkins we needed the following things:

1. A running Jenkins server
2. *Performance Plugin* for Jenkins
3. An executable Maven test project

We installed Jenkins and the *Performance Plugin* (Table 6.3.1) to be able to display the performance test results in Jenkins. Our test project was developed as a Maven project with modified JUnit test cases. The great advantage of these two performance tools is that they can be used together in a single build. JMeter could also be used with the *Performance Plugin*, but we seperated them from each other because of the testing level (Table 6.1).

The integration of both tools, JUnitPerf and ContiPerf2, is very simple. We had to install the *Performance Plugin* for Jenkins from the integrated plugin page. After we restarted the Jenkins server, we could add a post-build action to the project. Figure 6.33 shows the configuration posibillities of the post-build action. We used an error threshold for each build of 10 for unstable and 25 for failed, so we could see if more then 10 or 25 errors occured. With the input field for the report files we could also ignore some test files for our performance overview.

Figure 6.34 shows the response time and the error percentage of the ContiPerf2 test. The response time average response time of our test was about 7.3 seconds. We defined that the test failed if the response time is over 7.5 seconds. As seen in this figure it is only possible to receive true or false results. Either the response time is below 7.5 seconds and the test passes, or it is above 7.5 and the test fails.

With JUnitPerf we had a better view of the performance results of our load test. As seen in Figure 6.35 the percentage of errors is not only 1 or 0. We defined a timeout of 18ms for each test execution. We ran 2500 test exectutions and an average of 120 errors occured.

We also defined a throughput test for JUnitPerf (Figure 6.36). All the tests had to be finished in a overall time of 8000ms. The left side shows the average execution time of each test and the right side shows if the overall time of 8000ms was exceeded.

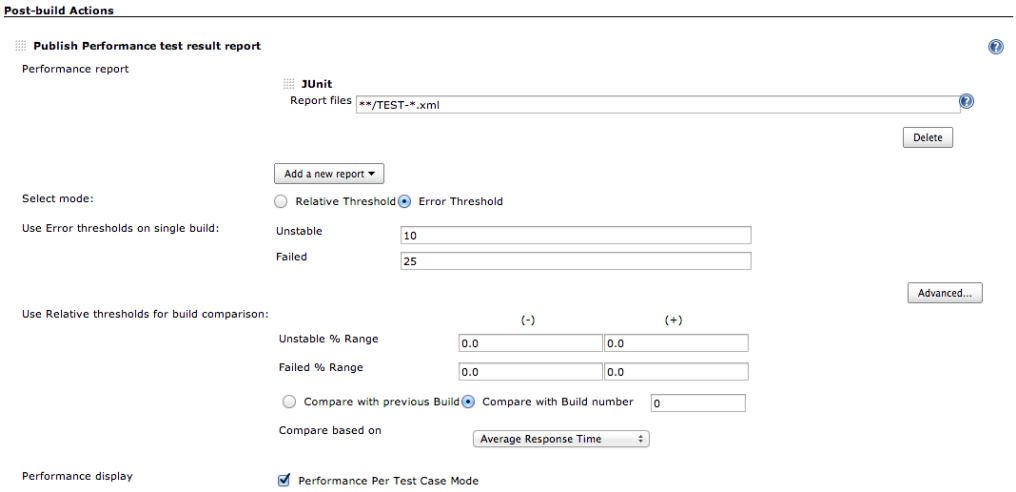


Figure 6.33. Post-build action settings

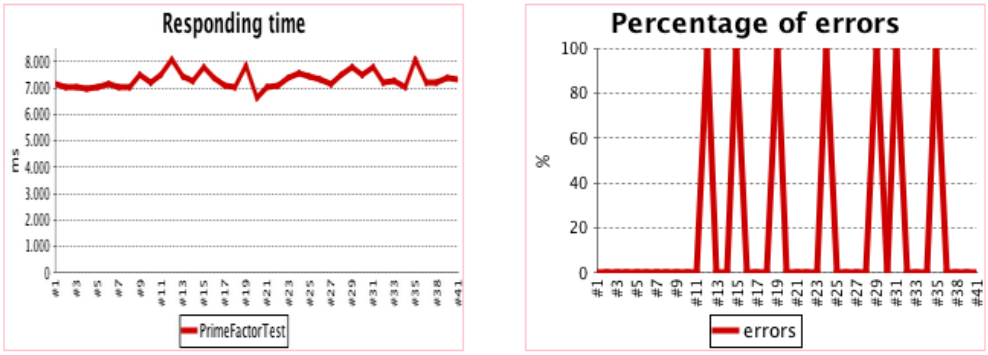


Figure 6.34. ContiPerf 2 - Performance overview

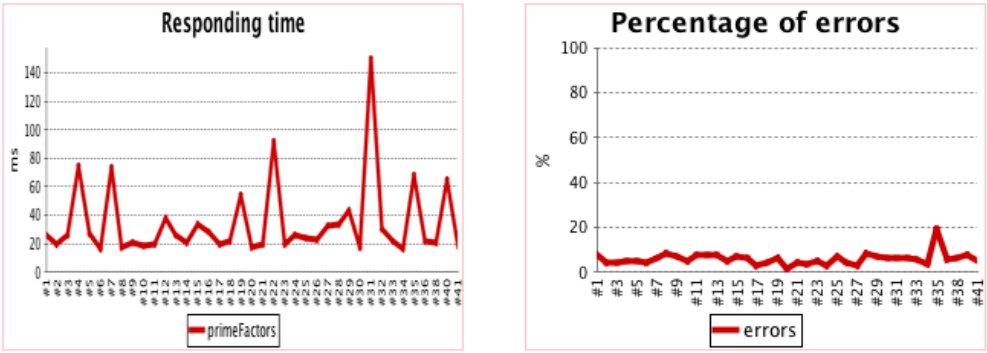


Figure 6.35. JUnitPerf - Performance overview (load test)

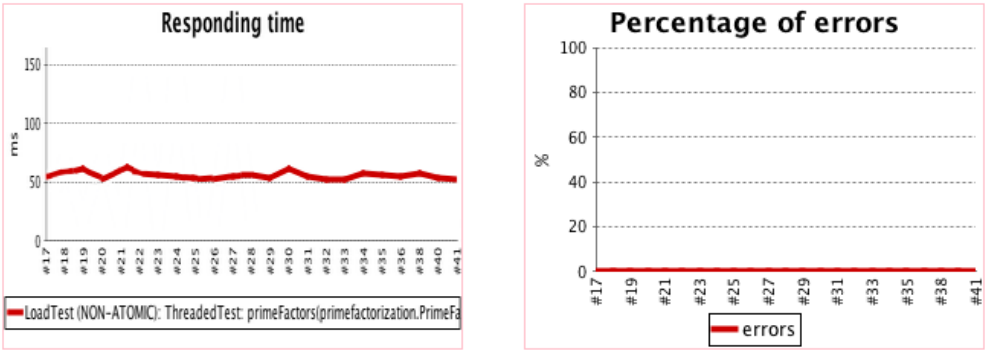


Figure 6.36. JUnitPerf - Performance overview (throughput test)

6.3.4 JMeter

Although JMeter is a GUI application, there are at least three possible options to run JMeter in a non-GUI modus:

1. Ant task
2. Command line non-GUI launch option
3. Maven plugin

We decided to go the Maven plugin way. It exists already a useable JMeter Maven project [3] that we used in this integration. We extended the test plan so that we tested a dynamic PHP page (1), a cached dynamic PHP page (2), and a dynamic PHP page (3) with a database connection.

Atlassian Bamboo

We found an add-on called ''JMeter Aggregator for Bamboo''. It aggregates the JMeter test results and allows reporting on the aggregated results across all builds. The installation of this add-on was not easy. First, the add-on is not officially supported for Bamboo and the last version of the tool works only with Bamboo 5.0 from July 2013. In addition, the add-on cannot be installed directly from the marketplace. Because of that, we needed to download the add-on from the website and move it to the Bamboo library folder by ourselves.

The configuration is rather easy. After the build-project and the corresponding job are created, the add-on can be activated in the job configuration at the tab ''Miscellaneous''. It is also possible to configure assertions, so that a build can fail, if the given assertion is not satisfied as seen in Figure 6.37. The add-on provides a form (Figure 6.38) that let us generate graphs. But the given metrics are not clear because most entries contain the word values. It is not clear what values should mean in this context.

Jenkins CI

With the performance test plugin, it is very easy to integrate JMeter into Jenkins. The plugin can be installed directly from the Jenkins web interface. The configuration and options are similar to ContiPerf 2 and JUnitPerf as seen in Figure 6.33. This plugin provides only two metrics, namely the response time and percentage of errors. Figure 6.39 shows a response time graph from the aggregated JMeter results and an error graph across the last 20 builds. It is also possible to get detailed results per build and page presented as a table (Figure 6.40).

[3] https://github.com/mlex/jmeter-maven-example

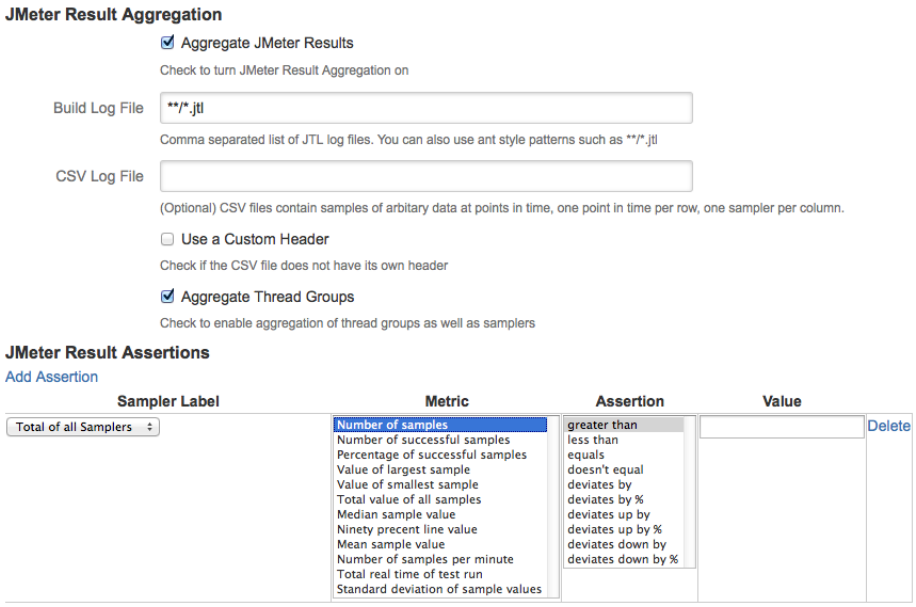


Figure 6.37. JMeter configuration in Bamboo

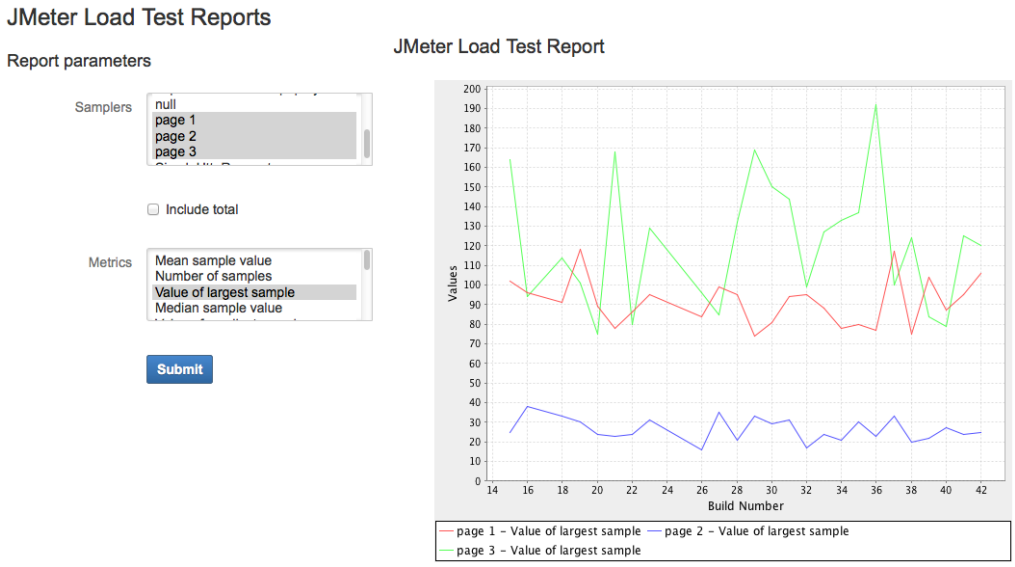


Figure 6.38. JMeter build results graph in Bamboo

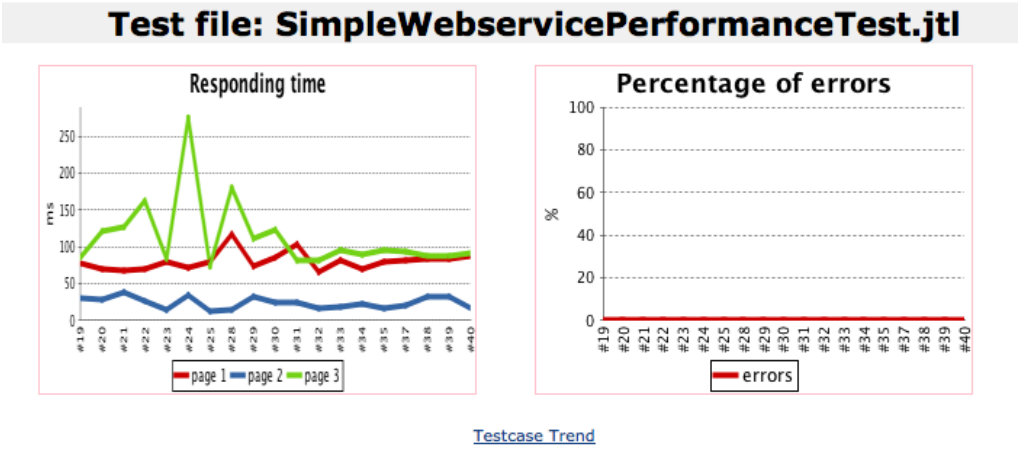
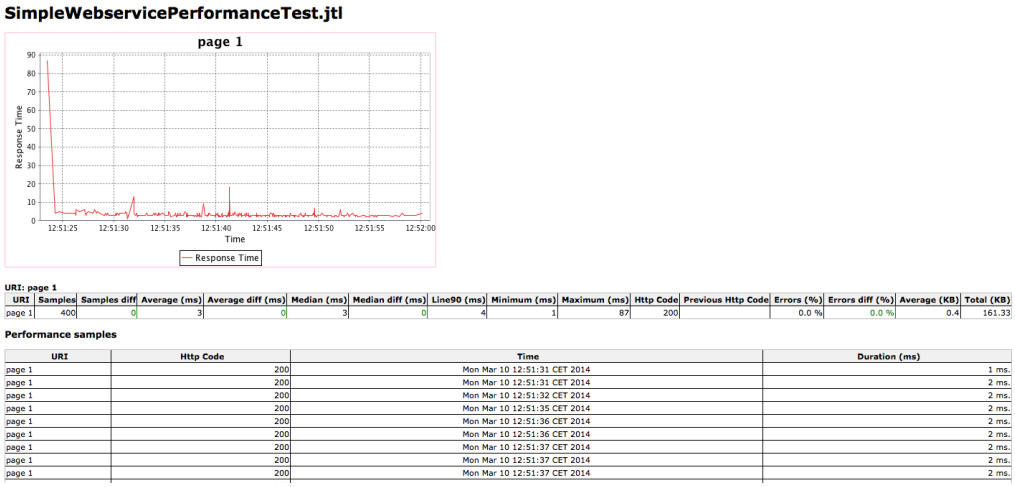


Figure 6.39. JMeter build results graph in Jenkins



URI	Samples	Samples diff	Average (ms)	Average diff (ms)	Median (ms)	Median diff (ms)	Line90 (ms)	Minimum (ms)	Maximum (ms)	Http Code	Previous Http Code	Errors (%)	Errors diff (%)	Average (KB)	Total (KB)
page 1	400	0	3	0	3	0	4	1	87	200		0.0 %	0.0 %	0.4	161.33

URI	Http Code	Time	Duration (ms)
page 1	200	Mon Mar 10 12:51:31 CET 2014	1 ms.
page 1	200	Mon Mar 10 12:51:31 CET 2014	2 ms.
page 1	200	Mon Mar 10 12:51:32 CET 2014	2 ms.
page 1	200	Mon Mar 10 12:51:35 CET 2014	2 ms.
page 1	200	Mon Mar 10 12:51:36 CET 2014	2 ms.
page 1	200	Mon Mar 10 12:51:36 CET 2014	2 ms.
page 1	200	Mon Mar 10 12:51:37 CET 2014	2 ms.
page 1	200	Mon Mar 10 12:51:37 CET 2014	2 ms.
page 1	200	Mon Mar 10 12:51:37 CET 2014	2 ms.

Figure 6.40. Detailed JMeter results in Jenkins

6.4 Results

In this section we summarize the results of our evaluation about the performance testing tools in context with the criteria catalog from Chapter 5. For each performance testing tool and CI server there is a table where each criteria from the criteria catalog is rated. 18 points are the maximum value a performance testing tool can reach.

6.4.1 Atlassian Bamboo: Blitz

Criteria		**Comments**
Documentation	$\odot$	An installation description exists.
Community	$\odot$	A forum exists. Questions will be answered within one day.
Integration	$\oplus$	Simple integration through the add-on manager.
Learnability	$\oplus$	Easy to learn. Simple rush could be executed within 10 minutes.
Likeability	$\odot$	Product is okay to use but the results could be displayed better.
Maintenance	$\ominus$	Development is behind. Update add-on for the compability with the newest Atlassian Bamboo is under development.
Software Cost	$\oplus$	The add-on is open source.
Support	$\oplus$	It exists a support forum. Questions will be answered within one day.
Usability	$\oplus$	Easy to use.

Overall, the Blitz add-on for Atlassian Bamboo receives 13 of 18 points.

6.4.2 Jenkins CI: Blitz

Criteria		Comments
Documentation	⊙	An installation description exists.
Community	⊙	A forum exists. Questions will be answered within one day.
Integration	⊕	Simple integration through the plugin manager.
Learnability	⊕	Easy to learn. Simple rush could be executed within 10 minutes.
Likeability	⊙	Product is okay to use but the results could be displayed better.
Maintenance	⊕	Development is behind but the current version is integratable into the newest Jenkins version
Software Cost	⊕	The plugin is open source.
Support	⊕	A support forum exists. Questions will be answered within one day.
Usability	⊕	Easy to use.

Overall, the Blitz plugin for Jenkins CI receives 15 of 18 points.

6.4.3 Atlassian Bamboo: JMeter

Criteria		Comments
Documentation	⊖	A wrong installation description exists.
Community	⊕	Older blog posts and questions about the add-on can be found.
Integration	⊙	Integration was not easy. The add-on cannot be installed through the add-on manager.
Learnability	⊕	Easy to learn.
Likeability	⊙	The user experience is improvable.
Maintenance	⊖	The plugin is officially not supported.
Software Cost	⊕	The plugin is open source.
Support	⊕	Question can be asked via Atlassian Answers.
Usability	⊙	Easy to use, but graph form lets still some questions open.

Overall, the JMeter plugin for Bamboo receives 11 of 18 points.

6.4.4 Jenkins CI: ContiPerf 2, JUnitPerf, JMeter

We created only one table for ContiPerf 2, JUnitPerf and JMeter because all these tools use the same plugin in Jenkins, the *Performance Plugin*.

Criteria		**Comments**
Documentation	⊕	An easy and short manual exists + example
Community	⊙	There is an overview of Jira tickets.
Integration	⊕	Simple integration through the plugin manager.
Learnability	⊕	Easy to learn.
Likeability	⊙	The design is simple, it could provide more metrics.
Maintenance	⊕	A stable version exists and it is under continuous development.
Software Cost	⊕	It is free and open source.
Support	⊙	The developer answers the questions on the wiki page.
Usability	⊕	Simple and easy to use.

Overall, the *Performance Plugin* for Jenkins CI receives 15 of 18 points.

Chapter 7

Conclusion

7.1 Motivation and Summary

It is important to test performance of software on a regular basis. Small changes in code can have immense consequences for the performance of the software. A continuous integration (CI) server with performance testing tools is the fastest and most secure way to detect performance issues. At first we did some research and collected information about Jenkins, Bamboo and many different perfomance testing tools. With this market overview and the requirements by adesso AG we reduced the amount of performance testing tools which we evaluated in detail. Finally, we installed the CI servers and integrated the chosen performance testing tools. In the next section we present our results and give a recommendation.

7.2 Results

During our reading up phase we had the impression that Blitz and Bamboo will be the favourite combination of CI server and performance testing tool. After deeper research and testing our first impression turned out to be wrong. It turned out that the performance testing add-ons for Bamboo are outdated and/or not supported anymore. At that time, Blitz is only compatible with Bamboo 3.4.4 from February 2012 but a new version is in development. The product owner Mu Dynamics told us that they are planning a release for the latest version of Bamboo in March 2014. Although we tested Blitz on an older version of Bamboo, it reached 13 out of 18 points and it seems to be a promising add-on for Bamboo. The JMeter add-on is outdated and worse, it is officially not supported anymore. At that time the latest release of the JMeter add-on works only with Bamboo 5.0. This can be seen by the fact that it is barely documented and the installation was complex with misleading installation instructions. The JMeter report form provides multiple metrics to choose from but the displayed graph is difficult to interpret because of insufficient axis labels. This is the reason it only reached 10 out of 18 points. For ContiPerf 2 and JUnitPerf we could not find a way to integrate them into Bamboo.

The performance testing plugins for Jenkins were all easy to integrate via the plugin manager into the current version of Jenkins. The plugin for Blitz is not explicitly developed for the latest version of Jenkins but it works like a charm. Blitz for Jenkins is easy to use and

self-describing. The only negative issue is that the results of the tests could be displayed more appealingly. Overall Blitz for Jenkins reached 15 out of 18 points. For Jenkins there exists one performance plugin which can build reports from the JUnit or JMeter report format. As a result of this the three plugins ContiPerf 2, JUnitPerf, and JMeter can be used together at the same time. Furthermore it is easy to extend the performance plugin by using a performance testing tool that supports the JUnit or JMeter report format. The only constraint is that it just supports the two metrics “response time” and “percentage of errors”.

All in all we are of the opinion that Jenkins had the better performance testing tools and better support. For websites in production mode we recommend Jenkins with the Blitz plugin because it can test a website under real conditions. If the website is under development, Jenkins with JMeter is the better choice because it can be used in private networks. Under the condition that unit tests should be used as performance tests we recommend Jenkins with the performance testing tool JUnitPerf. Unlike ContiPerf 2 test executions can fail partial.

Chapter 8

Appendix

LOAD TEST REPORT

DATE: 2/17/2014

TEST FROM : VIRGINIA

Query URL: http://www.st-softwaretechnik.de:80

Started at: Mon Feb 17 2014, 08:33:50 +01:00

Finished at: Mon Feb 17 2014, 08:33:50 +01:00

ANALYSIS

This rush generated **1,386** successful hits in **60 seconds** and we transferred **1.73 MB** of data in and out of your app. The average hit rate of **23/second** translates to about **1,995,840** hits/day.

The average response time was **399 ms**.

You've got bigger problems, though: **71.63%** of the users during this **rush** experienced timeouts or errors!

RESPONSE TIMES	TEST CONFIGURATION	OTHER STATS
FASTEST: 199 MS	REGION: VIRGINIA	AVG. HITS: 23 /SEC
SLOWEST: 785 MS	DURATION: 60 SECONDS	DATA TRANSFERED: 1.73 MB
AVERAGE: 399 MS	LOAD: 1-250 USERS	

HITS 28.37% (1386)
ERRORS 31.91% (1559)
TIMEOUTS 39.73% (1941)

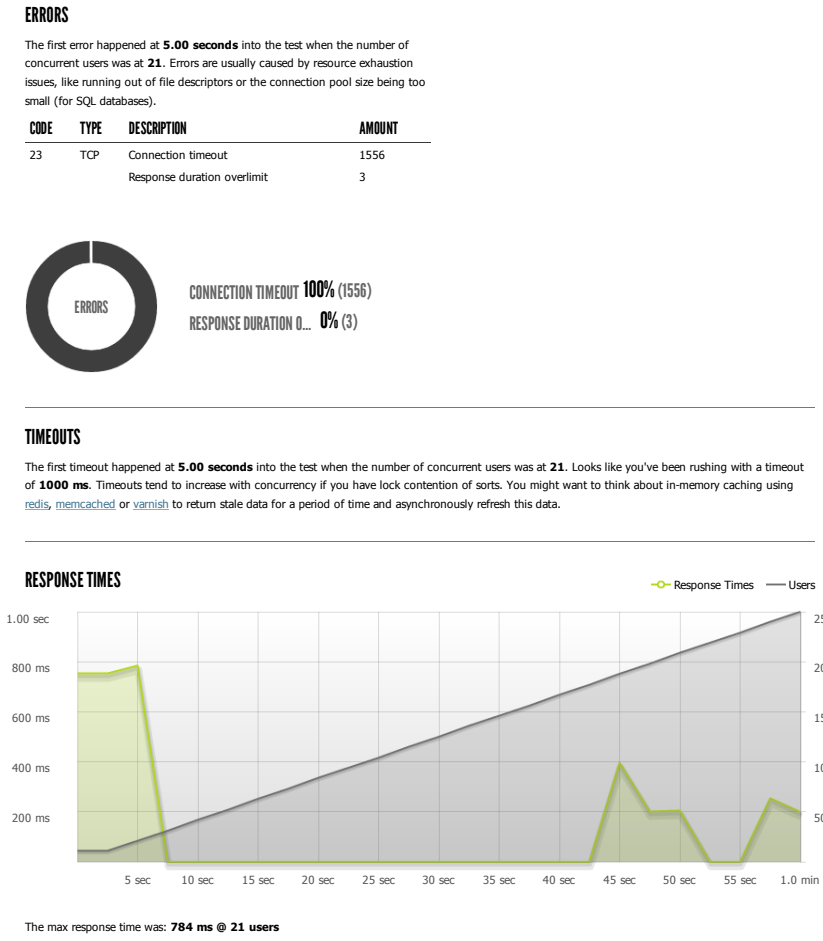
HITS

This rush generated **1,386** successful hits. The number of hits includes all the responses listed below. For example, if you only want **HTTP 200 OK** responses to count as Hits, then you can specify **--status 200** in your rush.

CODE	TYPE	DESCRIPTION	AMOUNT
200	HTTP	OK	7
503	HTTP	Service Unavailable	1379

HTTP 200 OK 1% (7)
HTTP 503 SERVICE UNA... 99% (1379)

Figure 1. Blitz response page one



ERRORS

The first error happened at **5.00 seconds** into the test when the number of concurrent users was at **21**. Errors are usually caused by resource exhaustion issues, like running out of file descriptors or the connection pool size being too small (for SQL databases).

CODE	TYPE	DESCRIPTION	AMOUNT
23	TCP	Connection timeout	1556
		Response duration overlimit	3

TIMEOUTS

The first timeout happened at **5.00 seconds** into the test when the number of concurrent users was at **21**. Looks like you've been rushing with a timeout of **1000 ms**. Timeouts tend to increase with concurrency if you have lock contention of sorts. You might want to think about in-memory caching using redis, memcached or varnish to return stale data for a period of time and asynchronously refresh this data.

RESPONSE TIMES

The max response time was: **784 ms @ 21 users**

Figure 2. Blitz response page two

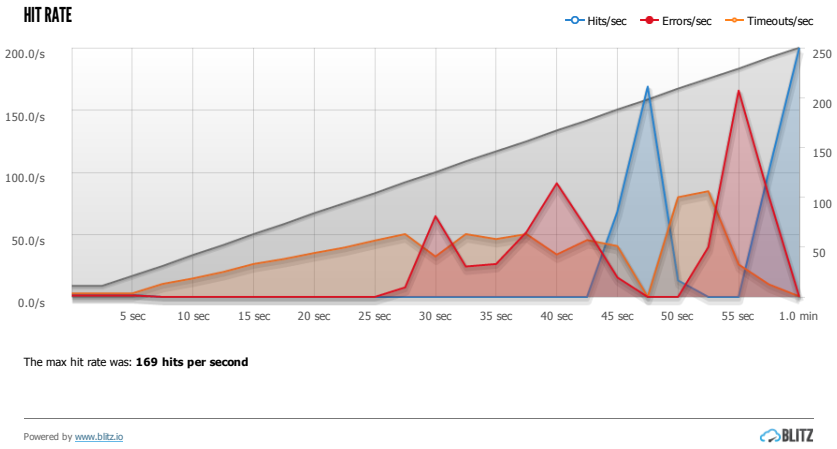


Figure 3. Blitz response page three

Bibliography

[*Apache JMeter*] Apache Software Foundation. Apache JMeter. http://jmeter.apache.org. Accessed: 2014-02-16. (Cited on page 25)

[*Jenkins Performance Plugin*] M. Carrasco. Jenkins Performance Plugin. https://wiki.jenkins-ci.org/display/JENKINS/Performance+Plugin. Accessed: 2014-02-16. (Cited on page 31)

[Duvall et al. 2007] P. Duvall, S. Matyas, and A. Glover. Continuous Integration: Improving Software Quality and Reducing Risk. Boston, 2007. (Cited on pages 1 and 3)

[Feuste and Schluff 2012] B. Feuste and S. Schluff. Continuous Integration in Zeiten agiler Programmierung. Feb. 2012. URL: http://www.heise.de/developer/artikel/Continuous-Integration-in-Zeiten-agiler-Programmierung-1427092.html. (Cited on pages 3, 4)

[Fowler 2006] M. Fowler. Continuous Integration. May 2006. URL: http://martinfowler.com/articles/continuousIntegration.html. (Cited on page 3)

[*JUnitPerf*]. JUnitPerf. URL: http://www.clarkware.com/software/JUnitPerf.html. (Cited on page 27)

[Meier et al. 2007] J. Meier, C Ferre, P Bansode, and S Barber. Types pf Performance Testing. Sept. 2007. URL: http://msdn.microsoft.com/en-us/library/bb924357.aspxl. (Cited on page 5)

[*ScalaMeter, Automate your performance testing today.*] ScalaMeter. ScalaMeter, Automate your performance testing today. http://axel22.github.io/scalameter. Accessed: 2014-02-16. (Cited on pages 28, 29)

[*Prime Factorization - Algorithm in Java*] L. Vogel. Prime Factorization - Algorithm in Java. http://www.vogella.com/tutorials/JavaAlgorithmsPrimeFactorization/article.html. Accessed: 2014-02-16. (Cited on page 20)

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben.
Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet.
Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens.
Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht.
Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Unterschrift:

Declaration

I hereby declare that the work presented in this thesis is entirely my own.
I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations.
Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before.
The electronic copy is consistent with all submitted copies.

Signature: