hochschule mannheim

Performance Comparison between JAVA and Node.js in terms of Scaling in Cloud

Zheng Zeng

Bachelor Thesis

for the acquisition of the academic degree Bachelor of Science (B.Sc.)

Course of Studies: Computer Science

Department of Computer Science

University of Applied Sciences Mannheim

11.11.2015

Tutors

Prof. Peter Knauber, Hochschule Mannheim

Jens Keller, SAP SE

Zeng, Zheng:
Performance Comparison between JAVA and Node.js in terms of Scaling in Cloud / Zheng Zeng. –
Bachelor Thesis, Mannheim: University of Applied Sciences Mannheim, 2015. **??** pages.

Zeng, Zheng:
/ Zheng Zeng. –
Bachelor-Thesis, Mannheim: Hochschule Mannheim, 2015. **??** Seiten.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich bin damit einverstanden, dass meine Arbeit veröffentlicht wird, d. h. dass die Arbeit elektronisch gespeichert, in andere Formate konvertiert, auf den Servern der Hochschule Mannheim öffentlich zugänglich gemacht und über das Internet verbreitet werden darf.

Mannheim, 11.11.2015

Zheng Zeng

Abstract

This project aims to evaluate the performance

Performance Comparison between JAVA and Node.js in terms of Scaling in Cloud

Contents

Chapter 1

Introduction

The promise of cloud has ushered the software engineering into a new era. Technology is to be used anytime and anywhere to untangle issues, provide solutions, bring people together and change their ways of living. This new era of net-centric web-based applications redefines how software is delivered to a customer and how the customer uses the delivered software. The digitization of economy impacts everyone. New businesses and leaders are emerging from nowhere. Companies appear and disappear much faster than ever before.

SAP , a company which has been renown for its on premise ERP solutions, also faces the pressure from customers to reduce TCO and increase agility. It has since long announced its cloud strategy to help customer master the digital economy. The reach of SAP systems is to be extended through cloud based access, ultimately reaching everybody everywhere in entirely new ways. Cloud Computing allows on demand software provisioning with Zero-Installation and automatic configuration at low cost and immediate access in ultra-scalable data centers, which leads to the next generation networked solutions. In the process of leverage cloud infrastructure to increase the business agility and to lower the TCO of customers, ultimately new types of applications are enabled.

Cloud solutions come with new capabilities and technical challenges for cloud software development. Different cloud software development platforms and frameworks that applied with different programming languages are used inside SAP. JAVA EE and Node.JS are the most prominent ones. There have been on-going discussions about how to take a choice between them for SAP applications considering the SAP environment. This paper aims to discuss this issue in terms of their respective performance and scalibility. The reason is simple. Among all the major

advantages brought by the cloud paradigm, scalability is the one that makes cloud computing different to an "advanced outsourcing" solution. Performance, which heavily affects customer satisfaction is a key metric.

Chapter 2

Related Work

Chapter 3

Technological Background

Chapter 4

Experimental Environments

4.1 Server running environment - Cloud Foundry

4.1.1 Cloud Foundry at SAP

4.1.2 PaaS - Service plans, Backing services and more

4.1.3 Limitations

4.2 Client running environments

4.2.1 Influential factors for measuring

4.2.2 Client in local machine

4.2.3 Client in Cloud Foundry

4.2.4 Client in Monsoon

Chapter 5

Testing Tool

5.1 Load generator - a self implemented scalable client

5.1.1 Design and function of load generator

5.1.2 Limitations

Chapter 6

Measuring Tool

6.1 Response time

6.1.1 Recording with Load Generator and its limitations

6.1.2 Retrieving data from ELK and its limitations

6.2 CPU and Memory consumption

6.2.1 Cloud Foundry CLI and its limitations

6.2.2 A self-implemented Ruby application and its limitations

Chapter 7

Test Scenario

7.1 IO Intensive

Store the current order of product and its meta information, find the which shelf stores the current requested product, assign an idling logistic unit to pick up the itinerary ... I/O operations make up the majority of SAP business scenarios. [TODO: DO I HAVE ANY STATISTIC TO SUPPORT THIS?]. In the I/O test conducted in this paper, applications are built to realize a scenario: advertisements are published in a bulletin board and clients can browse through the items.

The PostgreSQL backing service from Cloud Foundry is used as database. As the goal is to test how the application handles large amount of concurrency instead of the database efficiency, data will not be queried in high quantity or complicated joined actions in the test.

To bring Java and node.js to a comparable level, the applications are implemented with minimum use of frameworks so that the overhead or any other influence on performance can be first taken off the table.

Figure [TODO] shows the implementation of the Java application. It uses embedded Jetty server to handle plain HTTP requests. No REST or any other kind of web services is implemented. The connection with database is plain JDBC. CRUD operations are implemented with query statements. No ORM is utilized. Since the data structure is intentionally kept simple: only one table and with no complex data types, the application without ORM does not produce significantly more boiler codes.

Figure[TODO] shows the implementation of the node application. It uses the standard node.js library to build the web platform. One other dependency it has is the database connection.

Another library both applications have used is the logging library. The logging library is configured so it can be filtered in elastic search. Because the convenience provided by the existing cloud foundry ELK platform, it is able to log information relevant to single requests and retrieve them through elastic search. This function is used later when comparing the measuring results and determining the bottleneck.

7.2 Compute Intensive

Chapter 8

Test Results and Analysis

8.1 IO Intensive

8.2 Compute Intensive

Chapter 9

Conclusion

List of Tables

List of Figures

Listings