

TÓPICOS AVANÇADOS DE SISTEMAS DISTRIBUIDOS ARTICLE

JOÃO CALHAU, M36764



UNIVERSIDADE
DE ÉVORA

CONTENTS

| | | |
|-----|-------------------------------------|----|
| 1 | Abstract | 3 |
| 2 | Introduction | 4 |
| 3 | Architecture | 5 |
| 3.1 | Distributed Hash Table | 5 |
| 3.2 | Grizzly, Jersey and Maven | 7 |
| 3.3 | Tomcat | 7 |
| 4 | Evaluation | 8 |
| 5 | Comparison | 10 |
| 6 | Conclusions & future work | 11 |
| 7 | Bibliography | 12 |

1 ABSTRACT

Having worked with tomcat before, but never with Jersey, Grizzly or Maven. I set out to experiment joining the two together to give birth to something new that I had never seen before. A RESTful Web Application used as a name Look Up Service, implemented with a Jersey/Grizzly Server where a (so called) DHT would be used to store said names. And Also the implementation of a Web Application in tomcat, a brief explanation of what is a DHT, Jersey/Grizzly, Maven, tomcat.

2 INTRODUCTION

I tasked myself with a name look up service that was going to be hosted on a REST server, which would be able to communicate with a RESTful client, which in turn was to be used in a Web App.

The REST server is also where the data would be stored. The data being all the names that would be, or where already, stored. What was said to be used as a storage device was a Distributed Hash Table, or DHT for short. Seeing has the language in which I was going to try and build all of this was Java, I settled with a server using Jersey, Grizzly and Maven. On the client side of things I settled for a Tomcat hosted web service.

3 ARCHITECTURE

3.1 Distributed Hash Table

The DHT architecture I choose for this work was a Chord DHT, that basically is a peer-to-peer distributed hash table. that stores key-value pairs by assigning keys to different computers (or nodes). A node will store the values for all the keys for which it is responsible. Chord specifies how keys are assigned to nodes, and how a node can discover the value for a given key by first locating the node responsible for that key.

Using the Chord lookup protocol, nodes and keys are arranged in an identifier circle that has at most 2^m nodes, ranging from 0 to $2^m - 1$ (m should be large enough to avoid collisions). Some of the nodes will map to machines or keys while other will be empty.

Each node has a successor and a predecessor. The successor to a node is the next node in the identifier circle in a clockwise direction. The predecessor is the counter-clockwise.

The concept of successor can be used for keys as well. The successor node of a key k is the first node whose ID equals to k or follows k in the identifier circle. Every key is assigned to (stored at) its successor node, so looking up a key is to query $\text{successor}(k)$.

since the successor (or predecessor) of a node may disappear from the network (because of failure or departure), each node records a whole segment of the circle adjacent to it, the r nodes preceding it and the r nodes following it. This list results in a high probability that a node is able to correctly locate its successor or predecessor, even if the network is question suffers from a high failure rate.

The core usage of the Chord protocol is to query a key from a client (generally a node). The basic approach is to pass the query to a node's successor, if it cannot find the key locally. This will lead to a $O(N)$ query time where N is the number of machines.

To avoid the linear search ($O(n)$) Chord implements a faster search method by requiring each node to keep a finger table containing up to m entries (m is the number of bit in the hash key). The first entry of the finger table is the node's immediate successor. Every time a node wants to look up a key k , it will pass the query to the closest successor or predecessor of k in its finger table, until a node finds out the key is stored in its immediate successor. With this method the number of nodes that must be contacted to find a successor in an N -node network is $O(\log N)$.

Whenever a new node joins, three invariants should be maintained: each node's successor points to its immediate successor correctly, each key is stored in $\text{successor}(k)$ and each node's finger table should be correct. The best method is to initialize the finger table from its immediate neighbors and make some updates, which in this case is $O(\log N)$.

| Failures and Replication: | |
|------------------------------|--|
| Potential uses | Explanation |
| Cooperative mirroring | A load balancing mechanism by a local network hosting information available to computer outside of a local network, allowing developers to balance the load between many computers |
| Time-shared storage | In a network, once a computer joins the network its available data is distributed throughout the network for retrieval when that computer disconnects from the network |
| Distributed indices | Retrieval of files over the network within a searchable database |
| scale combinational searches | Keys being candidate solutions to a problem and each key mapping to the node, or computer, that is responsible for evaluating them as a solution or not |

3.2 Grizzly, Jersey and Maven

Let's start by talking about Grizzly Web Server. Project Grizzly is a pure Java web service built using the NIO API. Grizzly's main use case is the web server component for the GlassFish application server. With Grizzly we can build scalable and robust servers using NIO as well as offering extended framework components (NIO or Non-Blocking I/O, is a collection of Java programming language APIs that offer features for intensive I/O operations).

Jersey RESTful Web Services framework is an open source, production quality framework for developing RESTful Web Services in Java. Jersey provides support for JAX-RS APIs and serves as a JAX-RS Reference Implementation. Jersey simplifies the development of RESTful Web services and their clients in Java in a standard and portable way.

But what is REST? Well, REST stands for Representational State Transfer, it's a software architecture style for creating scalable web services. RESTful systems communicate using the Hypertext Transfer Protocol (HTTP) using the same verbs (GET, POST, PUT, DELETE) that web browsers use to retrieve web pages and send data to remote servers. Finally, Maven is a build automation tool used primarily for Java projects. Maven addresses two aspects of building software: first, it describes how software is built, and second, its dependencies. Contrary to preceding tools like Ant, it uses conventions for the build procedure, and only exceptions need to be written down. A XML file describes the software project being built (pom.xml)

3.3 Tomcat

Apache Tomcat is an open-source Java Servlet Container. Tomcat implements several Java EE specifications including Java Servlet, JavaServer Pages, Java EL, and WebSocket and provides a pure Java HTTP web server environment in which Java code can run.

Java EE or Java Platform, Enterprise edition provides an API runtime environment for developing and running enterprise software, including network and web services.

A Java Servlet is a Java program that extends the capabilities of a server. Although servlets can respond to any types of requests, they most commonly implement applications hosted on web servers. They are most often used to process or store a Java Class in Java EE.

4 EVALUATION

When I set out to do this work I had no idea of the difficulties I was gonna have. I created the server (on grizzly, Jersey and Maven), created some classes that where gonna help me test the server. Instead of a DHT I used a HashMap (since the HashMaps on Java are already Serializable, and it's very much like a HashTable). I created a Test file to test if the server was working, experimented on inserting 3 names and retrieving them. Was successful:

```
-----
Building jersey-service 1.0-SNAPSHOT
-----

--- exec-maven-plugin:1.2.1:exec (default-cli) @ jersey-service ---
Jan 24, 2017 4:13:21 PM org.glassfish.grizzly.http.server.NetworkListener start
INFO: Started listener bound to [localhost:36764]
Jan 24, 2017 4:13:21 PM org.glassfish.grizzly.http.server.HttpServer start
INFO: [HttpServer] Started.
Jersey app started with WADL available at http://localhost:36764/tasd/application.wadl
Hit enter to stop it....
```

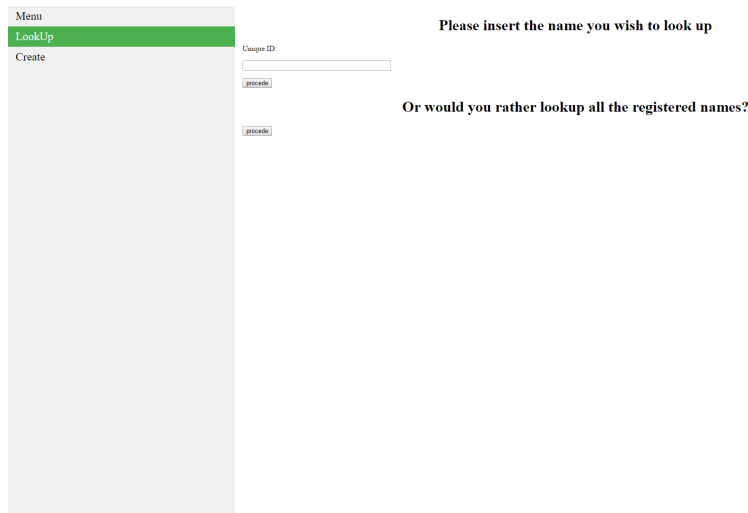
Figure 1: Server start up and run with Maven

```
-----
Building jersey-service 1.0-SNAPSHOT
-----

--- exec-maven-plugin:1.2.1:exec (default-cli) @ jersey-service ---
UniqueID: JDoe
  First Name: John
  Last Name: Doe
UniqueID: Uluthr3k
  First Name: João
  Last Name: Calhau
UniqueID: Flor
  First Name: Florbela
  Last Name: Espanca}
-----
BUILD SUCCESS
-----
Total time: 1.095s
Finished at: Tue Jan 24 16:12:53 GMT 2017
Final Memory: 6M/245M
-----
,
```

Figure 2: Test which inserted 3 names and retrieved them all

Then I went and tried to implement a Webservice based on Tomcat. I set up the index page, created a nifty cascade style sheet and configured the web.xml (all things I had done before):



Menu
LookUp
Create

Unique ID

proceed

Please insert the name you wish to look up

Or would you rather lookup all the registered names?

proceed

Figure 3: WebApplication running on tomcat

But when it was time to create a Servlet that used the Jersey Client to get what I needed from the HashMap, no matter how hard I tried, nothing seemed to work, and since there wasn't much online about using the two of them together I was at a loss. All that seemed to appear, every time I jumped to a Servlet page (all the html pages seemed fine), was this page:



HTTP Status 404 -

type Status report

message

description The requested resource is not available.

Apache Tomcat/8.5.11

Figure 4: "The requested resource is not available" error

All in all I wasn't able to complete my work, in practical terms, and I wasn't able to find any other work online that was similar to mine.

5 COMPARISON

I'm still a long ways from comparing my work to others seeing has my work isn't even half finished. Also I wasn't able to find almost anything online about someone with a work closely resembling mine. But If I had to say something about my work and how it could have been made in a different way is by changing the DHT structure, instead of a Chord structure I could have used a Tree-like structure like Pastry that each node maintains a "Leaf Set" (like Chord's successors, but bi-directional) and has pointers to $\log_2(n)$ nodes at each level i , Tapestry, Kademlia, Hypercubes that maintain pointers to a neighbor who differs in a one bit position and only has on possible neighbor in each direction but can route to receiver by changing any bit.

6 CONCLUSIONS & FUTURE WORK

All in all, I'd say this work wasn't what I was expecting in the beginning, but it turned out pretty interesting in the end, of course if I had managed to get past all those errors in the tomcat web application maybe I'd have a different idea. But the idea of a Chord DHT (based on a peer-to-peer network) is actually quite interesting and in the future I would like to take some time and build one so I can use, maybe not using tomcat but it's definitely something to consider. In the future I'd also like to take some time and search for different servers than tomcat, only tomcat because Grizzly, Jersey and Maven because all I had to do was type in the command line one thing and the whole server was set up, just like that. Tomcat however was a different story, to set it up I had to download the latest version of it on their website then I had to extract it somewhere, set up \$CATALINA_HOME so I could use it, after that I had to manually start it and manually deploy the webApp each time I had change something in it. Definitely something to avoid in my future work.

7 BIBLIOGRAPHY

- Carnegie Mellon School of Computer Science, Jeff Pang lecture on DHTs
- Creating a Basic REST Web Service using Grizzly, Jersey, and Maven
- Apache Maven
- Info on Apache Maven
- Non-Blocking I/O (Java)
- Apache Tomcat
- Info on Apache Tomcat
- Java Servlet
- Java EE
- Chord DHTs