

17CS352:Cloud Computing

Class Project: Rideshare

A HIGHLY SCALABLE MASSIVE PARALLEL RIDESHARING PLATFORM

Date of Evaluation: 15th May, 2020 11:00 AM

Evaluator(s): Prof. Venkatesh Prasad

Submission ID: XXXX (TBD)

Automated submission score: 10

|  |  |  |  |
| --- | --- | --- | --- |
| SNo | Name | USN | Class/Section |
| 1 | Hrishikesh V | PES1201700276 | VI-C |
| 2 | Varun R Gupta | PES1201700652 | VI-C |
| 3 | Ravendra Singh | PES1201700706 | VI-C |
| 4 | Aekansh Dixit | PES1201701808 | VI-C |

# Introduction

* Rideshare is a highly scalable massively parallel ridesharing platform that can be used to schedule rides that are shared between multiple users going along the same route at the same time.

## Related work

We have referred to the following sites to find the required documentation or instructions to get us started with the technologies used in this project:

* <https://zookeeper.apache.org/>
* <https://www.npmjs.com/package/node-zookeeper-client>
* <https://www.rabbitmq.com/documentation.html>
* <https://www.rabbitmq.com/tutorials/tutorial-one-java.html>
* <https://www.npmjs.com/package/amqp>
* <https://nodejs.org/en/docs/>
* <https://docs.docker.com/>
* <https://www.npmjs.com/package/dockerode>

## ALGORITHM/DESIGN

This is where you talk about either the algorithm that you used or the design of your system. How did you setup zookeeper.

ZooKeeper: Basically, after a lot of sifting and reading on the internet, we figured out that ZooKeeper is essentially a glorified always available and highly scalable network wide “tree” that can store information about our currently equipped and functioning containers. In addition to a regular tree, it has watchers and callbacks that make it extremely useful to keep track of running instances of our application without resorting to first hand inefficient methods such as polling or signals.

Our ZooKeeper setup consisted of the following: a root node “/”, having five children. We are only interested in three of them: “/election”, “/allNodes” and “/liveNodes”. The “/election” node is the primary node that we will be using to handle and manage the leader election process and keeping track of the nodes. Every dbServer container in our application starts off as a slave. Initially, two slaves are created. As soon as the container runs, it first tries to connect to the ZooKeeper server, and creates an EPHEMERAL SEQUENTIAL node representing itself under “/election”. Upon creating the node, it checks the list of all nodes to check for the oldest node. If it is the oldest node, it restarts itself as a master. Similarly, the second slave registers a node under “/election”, stores the oldest node as its leader and sets a watch on the election node. As soon as the current leader goes down, due to the watch, a callback is triggered, and all the slaves check to see if they are the oldest node in the server. The next oldest node now proceeds to become the master (by restarting itself) and the other slaves store it as its leader. Simultaneously, a new worker is automatically started upon the death of the master. This cycle repeats forever and ensure high-availability.

## TESTING

What were the testing challenges

Expand node:

Expand scaling;

Expand readDB:

Explain how you fixed the issues on automated submission

Again, expand the last few days…

## CHALLENGES

In ZooKeeper, one of the main issues we encountered is that the callback function was so complex and nested that it resulted in a lot of confusion over the variables and their scopes that are passed in and out of several nested callbacks, resulting in a callback hell. This was overcome by simplifying callback code by creating a clever recursion in the watch function, to automatically create a new watch and terminate the previous running instance of the function by returning.

## Contributions

Each individual team member must list their contributions towards it.

### Varun R. Gupta

1. Assignment 1: APIs 3, 4 and 5, AWS Setup and submission.
2. Final Project: High-availability (ZooKeeper), List Workers API, Master-Slave Crash APIs.

## CHECKLIST

|  |  |  |
| --- | --- | --- |
| SNo | Item | Status |
| 1. | Source code documented | ✔ ✔ ✔ ❌ |
| 2. | Source code uploaded to private github repository | ✔ ✔ ✔ ✔ |
| 3. | Instructions for building and running the code. Your code must be usable out of the box. | ✔ ✔ ✔ ✔ |