Yahoo! Cloud Serving Benchmark Web Interface visualisation

Internship Report

Author : Titouan Bion
Second year engineering student at the EISTI (Pau).

Addressed to Mr Peio Loubière & Mr Robert Krahn

July 25, 2016

Contents

1 Introduction

La chair de recherche utilise YCSB et d'autres logiciels de benchmark afin de benchmarker des BDD clouds (vérifier)

YCBS fournit un resultat à la fin Temps réel demandé Résultat text peu facile à manipuler Interface web de visualization

Enjeu : ne pas ralentir YCSB [A FAIRE :]

2 An university internship

The EISTI enable us to have three internships and I seized this opportunity to discover the more different working possibilities my field of study can offer. As I was considering a PhD. after my engineering school, I needed to know if I would have suited me.

I decided to use the EISTI partnership to achieve an intership at the Technische Universität Dresden in Germany. I chose this particular university because I heard positive feedbacks from acquaintances that studied there. I have been able to discover a research environment and discuss with doctors or PhD. students which nourished my thought and helped me make a choice for the next step of my career.

3 Internship environment

3.1 Technische Universität Dresden : global overview

[A FAIRE :]

3.2 System engineering chair

[A FAIRE :]

3.3 Personal observations and induction

[A FAIRE :]

mettre ici l'organisation du projet et le ressenti de l'environment

3.4 Working method

I used an AGILE method to develop this project. As I was working alone on the implementation side, I needed to confront my features quickly to future users in order to have feedbacks. My cycle was simple, I began with a strong implementation phase followed by a testing and debugging phase. Then I used my tools to evaluate my solution and judge if it was suitable according to the expectations and use cases.

For example, at the end of my first cycle, my application was working great for small datasets but when it needed to handle thousands of points it was crashing both on server and client side. So, I made a complete review of my code to find bottlenecks and fixed them one by one. I will technically develop these optimizations in section 4.4.

4 Building a solution

4.1 Exploration phase

API between YCSB and the NodeJS server At first, I wish to expose a RestAPI directly from the YCSB Java application. I wished to use technology like Hazelcast for example. Sadly, it was drastically slowing the warmup phase of YCSB so I tried a lightweight solution with ActiveJDBC and Spark Java (not to be confused with Apache Spark). Even this lightweight solution was solution YCSB too much. Also, it forces the user to let YCSB running in the back even if no benchmark was launch. The application modularity was not good enough for the maintenability of the solution.

I finally decided to let a MongoDB storage database be my API between the YCSB application and my web server. The database connection is quicker thus has a reduced impact on the warmup phase compared to the former solutions. Also, this modularity provided by the storage DB allows my client/server application to communicate with every software capable of storing data into a MongoDB database.

The web interface was a prerequisite to this project. Multiple chart library are available but I chose Highcharts library because of its display optimization. I will develop these optimizations later in this document.

YCSB points fetching routine As my solution needed not to slow YCSB, I made all storage operation in a different thread. The data was stored into other threads variables — YCSB client threads — and the way of fetching values was our top priority. Indeed, this operation is the one that impact YCSB the most.

I though of multiple approaches to retrieve YCSB data. First, I designed what I called a Database Hook that would launch a task in the storage thread at each measurement. As the first method was not efficient and slowed YCSB too much, I implemented a periodic fetching process that was handle by the storage thread. Again, I will elaborate on benefits of this process in the next section. [A FAIRE : *REF ?*]

4.2 Application architecture

4.2.1 Overall workflow

[A FAIRE : *Schema*]
[A FAIRE :]

4.2.2 Backend specifications

[A FAIRE :]

Fetching method [A FAIRE :] Executor not to disturb YCSB
Fail safe process and easy shutdown
concurrency map for handling problem

Communication with Storage DB [A FAIRE :] Native driver and BULK writes for efficiency

4.2.3 Frontend specifications

[A FAIRE :]

NodeJS Server [A FAIRE :] Lightweight, its modularity is very efficient with our small scale project Handle MongoDB and expose a RestAPI easily

Communication with Storage DB [A FAIRE :] Native NodeJS driver, mongoose was too slow and useless in our case (very simple requests and aggregation)

Handling multi millions benchmark [A FAIRE :] Mongo aggregation process

4.2.4 AngularJS Client

[A FAIRE :] Maintainable solution and structured client, updates and dynamic behaviour way simply handled, RestAPI request Encapsulated into services for a better use

Providing also Angular Material for an easy style and good looking client

Highcharts library [A FAIRE :] client aggregation process for view optimisation (Data-Grouping)

4.3 Optimizations

[A FAIRE :]

4.4 Increasing support

[A FAIRE :]

4.5 Performances

[A FAIRE :]

4.6 Limitations

[A FAIRE :]

5 Internship benefits

[A FAIRE :] As I said,

découvrir la recherche, intéressant mais environment trop scolaire pour moi et j'ai envie de changer d'air pour l'instant quitte à revenir après

6 Conclusion