(PRACTICAL) PROJECT PROPOSAL: A MAP-REDUCE FRAMEWORK

Soroosh Zare
soroosh.zare00@gmail.com

Farzin Nasiri
farzin-nasiri@hotmail.com

ABSTRACT

If you are reading this, you have probably heard about divide & conquer algorithms. Consider $\{P, M, R\}$, where P is an abstract problem, M is a function which solves problems P' in the same class of problems as P, and R is a function that merges the result of smaller sub-problems to achieve the solution for a bigger problem. M and R are usually called **Map** and **Reduce** functions from our point of view. Essentially map is a function used for transforming data and reduce is used for calculating a single answer from possibly many given answers for smaller problems. Now, given these 2 functions, we can use them to solve P even if they are black boxes, and we have no understanding of how they work. We can go further and divide the computation between different processes and provide them the function M, and finally use R on the gathered result to reach a final answer. The benefit of this solution is that the work can be done in parallel. The goal of this project is to implement a MapReduce framework built on top of this abstraction and do some experiments of different scenarios.

1 INTRODUCTION

The abstract above was an introduction to MapReduce. The goal is to implement a fault-tolerant MapReduce framework that runs on a cluster of nodes, possibly on different machines. We use the Actor model for implementing the framework. To handle process faults, we use links between a supervisor and other processes to take necessary measures when needed.

2 BASELINE PROJECT

First, we build a base project that can get map and reduce functions from the user, and given a data, run MapReduce on it, but on a single process.

We use map and reduce abstractions from the very beginning to make development of the next stages easier.

3 PROOF OF CONCEPT

As a proof of concept, we implement a MapReduce framework that runs on different processes, but yet doesn't handle process faults. This gives us some time to do initial experiments without the need to implement fault tolerance, which could take more time than anticipated.

4 FAULT TOLERANCE + HOW TO REACH IT

After the previous phase is done, we move on to implement fault tolerance. At the time of writing, we haven't decided on the exact algorithm to detect process faults. What we plan to do is design an infrastructure that uses supervisors to catch process faults, then run experiments on possible

candidates for the fault detection algorithm and finally choose the best algorithm based on some chosen metrics.

5 Final Goals & Evaluation

After the implementation is done, we test the framework on different problem domains and their respective map & reduce functions to see how it performs. Finally, we compare the performance in different network conditions, like delays, node failures rate, performance of each node, etc.

We expect to implement at least up to the Proof of Concept Project by date of project submission (we can't simulate node failures in this case), but the ideal scenario would be finishing the fault-tolerant design as well, where we could use node failures experiments as well.

6 Data & Technical Requirements

We wanted to use a functional programming language for this project, after researching about some languages, specifically scala and elixir, we decided to use elixir. This decision was made mainly because elixir has native support for most of the things we wanted to do, let's say message passing and support for actor model, without the need of any libraries. Meanwhile, if we wanted to use scala, we had to use libraries like Akka, which comes with the extra learning curve, and we didn't want to risk the little time we have on that.

For monitoring purposes, we use AppSignal, they kindly provided us an opensource sponsorship which lets us use their service for free.

We also need to have some datasets for the final experiments. At the time of writing, the decision is to choose problem domains where random data generation is feasible, and generate large random datasets based on our needs.

References

Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, pp. 137–150, San Francisco, CA, 2004.