Design and Implementation of Distributed Application: Project Report

Wallace Garbim
Instituto Superior Tecnico
wallace.garbim@tecnico.ulisboa.pt

Miguel Crespo
Instituto Superior Tecnico
miguel.crespo@tecnico.ulisboa.pt

Florian Ehrenstorfer
Instituto Superior Tecnico
fehrenstorfer@gmail.com

Abstract

In this paper we present a simplified function-as-a-service cloud platform, which enables the distributed execution of a chain of operators on a shared storage. The system consists of one scheduler and multiple worker and storage nodes. Workload is balanced equally between workers through the scheduler. Data consistency and fault tolerance are guaranteed by the storage nodes. We present the problem and motivation first, then describe our solution in detail, and evaluate it at last in this paper.

1. Introduction

Distributed function-as-a-service cloud platforms are increasing in popularity with big cloud providers like AWS and Google Cloud offering custom solutions. They offer great flexibility with of the shelf plug and play options as well as customizable and self-developed options. For this project we developed our own distributed function-as-a-service cloud platform, a simplified version of other platforms. We focused on its design, implementation and evaluation in the project in this paper.

2. Problem

The function-as-a-service cloud platform allows to run applications in a distributed network. Applications consist of a chain of operators, which can access a shared storage system. Because the storage system is distributed as well, one of the main problems is data consistency and fault tolerance. In order to maintain data consistency a gossip protocol is used to propagate updates through the network. The storage system supports three operations: read, write, and updateIfValueIs. Records are stored with the value and a version number. For the identification of the correct storage server to contact about a value, consistent hashing is used. For data consistency, operators have to read consistent version of the data, which has to be ensured by the system. Because the system also has to provide fault-tolerance, data has to be distributed between storage nodes, and reads have to be redirected. The different operators are assigned to workers by a scheduler, and then send from one worker to another after the operator has finished.

3. Solution

In the following section, we will describe our proposed solution.

Our system consists of five main processes: The scheduler, worker and storage nodes, and the Puppetmaster and PCS processes. The Puppetmaster is the main entry point of our system and controls the startup and workflow of the whole system. The PCS enables the Puppetmaster to startup the three different types of nodes remotely. Because all three nodes receive their configuration information as command line arguments at startup, the whole system could also be run without the Puppetmaster and PCS. The Puppetmaster can be run with single commands in the command line in a loop or by providing a full script, which will be executed by the Puppetmaster.
To run the Puppetmaster the following command has to be executed inside the Puppetmaster directory: *dotnet run.*
To run the Puppetmaster with a full script the following command has to be executed inside the Puppetmaster directory: *dotnet run ¡ scriptname.* scriptname has to include the path to the script.
The PCS process can be started with *dotnet run* inside the PCS directory. It only has to be started once to start multiple other nodes. All printed output from the started nodes will be printed in this console. The Puppetmaster

sends start up requests to the PCS, which then starts worker, storage or scheduler nodes as a new process. All other types of requests are also send from the Puppetmaster to PCS, where they are distributed to the relevant nodes.
The scheduler is responsible for the assignment of operators to workers and the distribution of workload between them. For that it stores the number of operators each worker currently works on and assigns the next operator to the worker with the fewest current operators. Because of that workers send completion confirmations to the scheduler when they have finish working on one operator.
Workers receive the chain of operators and execute the operator that was assigned to them, before storing the output in the chain and sending the chain to the next worker in line. Since they only receive the name of the operator, workers select the operator with reflection. For that the operator .dll file has to be placed inside the *Operator* folder inside the workers directory. The workers get the url of all storage nodes and the scheduler from the meta data of the chain.

4. Evaluation

5. Conclusion