INF-419 Principles of Distributed Systems
Class Project 2024

Vasilis Samoladas

May 5, 2024

Abstract

Project Topic: Implementation of Map-Reduce on Kubernetes

Groups: of 4 people

Due date: Friday, June 28 2024

Deliverables: All source code, plus a presentation.

Presentation dates: The 28th and 29th of June.

Extensions: due to the situation this year **no extension will be given**.

1 Overview

You will develop a distributed system for performing parallel computation, along the lines of Map-Reduce, as described in the paper

> **MapReduce: simplified data processing on large clusters**
>
> Authors: Jeffrey Dean and Sanjay Ghemawat
>
> In: Proceedings of Symposium on Operating Systems Design & Implementation (OSDI'04) Vol. 6, December 2004, Pages 10
>
> Full paper: `https://www.usenix.org/legacy/publications/library/proceedings/osdi04/tech/full_papers/dean/dean.pdf`
>
> CACM version: `https://dl.acm.org/doi/10.1145/1327452.1327492`

Note: you can also find the paper in **e-class**, in section "Lab (Εργαστήριο)"

1.1 System Concepts

User A human user who has the right to submit jobs.

Administrator A human user who has all rights to administer the sytem (create and delete users, configure workers etc)

Job A parallel computation executed by your map-reduce system.

Input format A *data format*, which can be read from a file, or many files, and converted into a collection of key-value pairs, suitable for input to the mappers.

An input format can be customized for each job, but you are only required to implement one. This can be, e.g., a JSON-based data format, or an AVRO-based data format.

Output format A *data format* which can be generated from a collection of key-value pairs, generated by the reducers, and written to a file (or to many files).

An output format can be customized for each job, but you are only required to implement one. This can be, e.g., a JSON-based data format, or an AVRO-based data format.

Mapper A function that can serve as the mapper of a map-reduce job. This can be given as a standalone function, or as an object.

Reducer A function that can serve as the reducer of a map-reduce job. This can be given as a standalone function, or as an object.

Worker A node (container) that can execute the necessary computation (mappers, reducers, shuffle) for completing a Job. Workers are able to execute code provided by each job.

Master A node (container) that can coordinate and monitor the proper execution of a job. Each job has one master. Note that the master node is also typically a worker node, but this is not required.

1.2 Requirements

1. The system will authenticate all **users**.

2. Each user must authenticate with a password, at an Authentication Service, which will issue a user token once. Subsequently, all other services in the system will use this token. New users shall be created by the Authentication Service and will have a unique user name.

3. Authenticated users may submit **jobs**, consisting of a pair of map/reduce functions, and a **filename**.

4. Users interact with the system via a **User Interface (UI) service**. The UI service should be accessed through a Command-Line Interface (CLI), implementing at least two commands:

   - A **jobs** command, which can submit new jobs, and view the status of existing jobs, and

- a **admin** command, which is used to administer the system (create and delete users, configure nodes, etc).

  Both of these commands can support sub-commands and options as needed.

  It is possible that these commands be implemented as an API in a scripting language (e.g. Python) and not as a CLI.

5. Job code (mapper and reducer) must be given to the system via well-defined interfaces. Depending on the language of implementation, you may use java class files, C/C++ source files, Python source files, etc.

6. To orchestrate your containers, you will use the **kubernetes** orchestrator. You can install kubernetes locally using `minikube`.

7. Each worker should execute in its own kubernetes pod. The image of this container should be prepared by you. You should consider the set of workers managed by a kubernetes `StatefulSet`.

8. Your system should support recovery on crash of workers or other nodes unexpectedly, via command. This facility can be used to check the ability of your implementation to recover after errors. As workers are Kubernetes pods,

   The simplest way to implement such a facility is by deleting the pod of the worker, via the kubernetes `kubectl delete` command.

9. In order to provide fault tolerance in the presence of crashing nodes, the system needs to maintain state data in a replicated, consistent data store, marked as **Distributed Data STore (DDS)**. You can use various possible alternatives for this, including

   - Apache Cassandra
   - etcd
   - zookeeper

10. You can build your project using any tools you want. However, the best candidates for finding help online, are probably Python or Java.

    Remember that ChatGPT is your friend in learning a new API/library.

2 Implementation

Each project team will consist of 4 members, and will implement their own set of services, following the micro-service-based 3-layer architecture.

An overview of all services can be seen in Fig. 1. A brief description of the services is as follows:

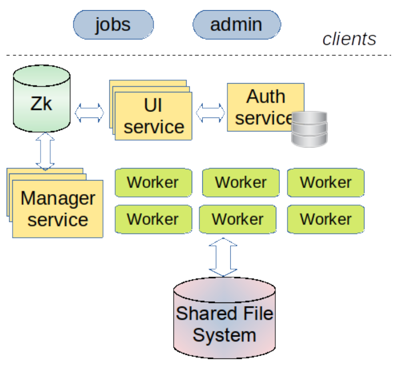

Figure 1: An overview of the services participating in the platform.

.

User Interface Service: This service is responsible for providing access to the map-reduce functions, via the two clients. It is replicated so that it can serve a high load of requests from clients.

This should be a stateless service. You can implement it in Kubernetes as a `Deployment`.

Authentication Service: it is responsible for maintaining the user authentication information and other relevant personal info (e.g., user email), as well as the roles that each user has (plain user or admin). This service maintains its own database, where it stores all user-related information.

It is not required this service to be especially sophisticated. A standalone pod is sufficient for this service, or a `StatefulService` with a `replicaCount` of 1. For the user database, a simple relational database can be utilized or you can re-use the DDS

Manager service: This is the main service of the whole system. It is an internal service that manages and monitors the execution of jobs.

To be resilient, this is a replicated service, that should probably be implemented as a `StatefulSet`. Each job is handled by one of the replica pods. If a replica fails, Kubernetes will replace the failed pod and the information in DDS can be used to continue managing the jobs of the failed replica.

When a new job is submitted to the system, the UI service selects one replica of the monitoring service and passes it the relevant information so that the new job can be scheduled for execution. You can use any balancing technique for this, for example a hashing scheme.

Distributed Data Service: This service is used to provide consistent, reliable and fault-tolerant data storage to the rest of the system. You do not need to implement this service, but you do need to design the information that will be stored on this service carefully.

Workers: These are the nodes where all execution happens. The system should (theoretically) scale to thousands of these nodes. When a worker fails, the task that was assigned to this worker must be re-assigned to another worker.

Shared File System: This is where input and output data can be stored. There are various options for this store, such as:

- A shared `PersistentVolume` created using attribute `ReadWriteMany` (we will discuss this in the lab).
- A storage service that you can set up independently, such as a (local) installation of MinIO (`https://min.io/`).

3 Deliverables

You should be prepared to deliver:

1. Full source code of all parts of the system (including Makefiles, Dockerfiles, Ant/Maven files, Kubernetes manifests etc). **Just delivering the project directory of some IDE such as Eclipse, Netbeans or IDEA, is not sufficient.**

   Ideally, one should be able to install your system on Kubernetes by running a simple script (but you may need to provide some assumptions on the container setup).

   Consider the idea of keeping everything in a common **git** repository (e.g., on github or gitlab). Using a public repository is fine with me, it is ok if teams see other teams' code.

2. A presentation (in Openoffice, Powerpoint or PDF) describing your implementation, including

   (a) Tools, frameworks and libraries you used

   (b) Choices that you made (e.g., how did you implement the web interface)

   (c) Execution on 8 nodes or more (we will use the *Okeanos services*, details will follow) and measure the performance of some indicative service calls under concurrency (use Apache Bench for this).

   (d) System recovery response, if a PlayMaster node fails.

3. You should be prepared to present the above presentation on the last week of classes at the end of the semester.

4 Advice

To properly implement your services and their functionality, you should use libraries and frameworks for providing major services. If you have already used such software, it is acceptable to use them for this project. However, a goal of this project for you should be to familiarize yourselves with new software, so do not hesitate to try learning new things.

To implement some parts of the whole system, you will need knowledge that we will study later in the course. So that you make continuous progress on the project, it is important to plan your coding and studying work, on those parts that are covered first.

Here is a short list of advice.

- First, form a team (4 people per team), exchange emails and other contact information. Agree on the tools to work with each other. This is extra important since you can start working without meeting each other.
  - A teleconferencing platform. Zoom is a good choice in this regard.
  - A platform where you can message each other quickly (email can be hard to use sometimes). I use `slack` and can recommend it.
  - A platform to exchange source code. If you are not on `github`, get on it now! Also, make sure you know how to use *git*
- Start selecting the software that you will use to implement your project. I strongly recommend one of the following "language ecosystem" choices:
  - Java ecosystem (including scala or groovy if you know or want to learn them) is of course an excellent choice. This is the preferred ecosystem for information systems around the world, and learning it better is a task that you will not regret.
  - Python is also an excellent choice (my personal favorite), and offers many excellent facilities for implementing our project. Python has recently become the #1 language for data analytics, machine learning and high-performance computing. You can't go wrong with it.
  - C++ is a good choice, if everyone in your team knows some C++. Otherwise, it may be a bit difficult to manage.

  Once you have selected the ecosystem, it is a good idea to also select a good build system for your code and get familiar with it.
- Start playing with `docker` (highly recommended) or any other platform for deploying containers (you are on your own!!). In particular, learn how to build a Docker image and how to deploy a swarm. There is excellent documentation on the Docker website.

- The next step is to design your implementation. Make sure you are fully aware of what the service API of each of the middle tier services is. Use UML a lot to specify your design, and then code your UML. If not, you will forget what your design is, and it will be hard to retrieve it from source code.

- Make sure that you write unit tests for every piece of code that you use. Unit testing is a very useful habit that you need to acquire as programmers, and it is going to save you 50% of the effort you will otherwise put in building your software.

  A very important use of unit testing is to test tools that you test the libraries and tools that you will use. Write small tests to use the tools the way you intend to use them in the code. These tests will make you more confident that you have understood the use of these tools, and are a great way to learn new APIs and libraries.

- Share the work. One of you should take charge for each new technology that you learn. For example, one can be in charge of docker and deployment, another can be in charge of API development. All of you should probably work on the middle tier services and the databases. What one learns, she transfers to the other.

Good luck, and remember:

KISS : Keep It Simple Stupid!
and
RTFM : Read The Fine Manual!