ID2221 - Reading Assignment 1

Authors: Miquel Larsson Corominas, Dmitry Siniukov

1 Summary

In this first reading assignment, two papers will be reviewed: "The Google File System", and "MapReduce Simplified Data Processing on Large Clusters", which have different motivations, but are related by the fact that they both deal with the scalability issues and problems that appear due to the complex nature of working with large amounts of data. From now on, the first paper will be referred to as the "GFS" paper, and the second one will be referred to as the "MapReduce" paper.

2 Motivation

As mentioned in the last section, both papers have different motivations, but share the core issue of dealing with large amounts of data. The GFS paper describes the necessity of designing a solid architecture capable of handling and storing the unfeasible quantity of data that Google requires, in a scalable and distributed way.

On the other hand, the MapReduce paper is focused on the need of creating a framework to easily encapsulate the natural complexity of parallel processing, making it easy for programmers to create and deploy these kinds of resources, for large data sets, in a highly scalable manner.

Furthermore, both papers put a lot of emphasis in the need of deploying the solutions on inexpensive commodity hardware, in order to reduce costs and required investment on infrastructure. Moreover, they both put an extra attention to the need of providing fault tolerance, due to the unavoidable nature of component failures and any other sort of human or software bugs.

3 Contributions

Regarding the GFS paper, the main contribution (speaking on a high level) was to develop a highly scalable and a fault tolerant file system that would satisfy the needs of Google itself. However, the paper's contributions also include solving some problems that are quite generic and can be related to data processing tasks that are common to other settings as well.

One of the generic problems is, having "inexpensive commodity hardware", to ensure the availability, safety, and congruence of the data, treating failures as a normal situation.

Another problem is related to the files' size. Rather than massively using several-KB files, they operate with large (multi-GB) files in most cases. So, some parameters of the system (like I/O and block sizes) had to be reconsidered.

Lastly, most of file updates are appends rather than changes of existing data. The latter also puts some limitations and assumptions on the file system to be developed.

Then, regarding the second paper; The whole idea was inspired by the "map" and "reduce" primitives that were already used in functional programming. Due to the nature of the problems Google were dealing with at the time, these primitives seemed to be a natural fit, which then led to being able to creating a whole framework and programming model that enabled automatic and simple parallelization and computational distribution of large scale problems that achieved high performances on commodity hardware, with a good fault tolerance.

4 Solution

In short, the main GFS design decision is using a centralized master server and multiple

(replicable) chunk-servers. So, to get a chunk of data, a client sends the first request to the master to get the correct chunk location. After that, the client is able to "ask" the correct chunkserver to retrieve the actual data. By using large chunk sizes and chunk leases, giving priority to the primary replica in data updates, the master's load is minimized.

To address the problem of parallel appends, GFS has introduced an atomic operation called "record append", in which a client needs to send only data (without offset). Then, the system appends it atomically and returns the offset to the client.

The whole MapReduce solution is too complex to summarize in this report, but a short summary will be given:

- Map function is applied by workers to the input data, being the output buffered into temporal memory. A master process is in charge of distributing this data among the available workers.
- The master forwards the location of the buffered output to the reduce workers, which read the data and stores it by the map output keys, so all same keys are grouped together.
- Then the reduce workers iterate over the data and passes the unique keys with their values to the reduce function, whose output is then appended in a final output.

Additionally, fault tolerance is an important component of the framework design. From the worker point of view, each worker is pinged periodically, in order to verify that they haven't failed. Any failed map tasks are then re-executed by any available worker, and the reduce tasks are notified of this change. This simple strategy makes the system resistant to large-scale worker failures. In the case of the master process, failure aborts the whole process, as the possibility of the master failing (being a single process) is extremely low.

5 Opinion

Regarding the GFS paper:

- In our opinion, one good decision Google took when defining the requirements and needs of the system, was considering the faults and component failures as a norm, as they are unavoidable, instead of as an exception. This point of view made data integrity management a core need of the system, which derived in an innovative repair system which automatically verifies and checks that the data chunks are available and correctly distributed in the system.
- The complexity of a system like this is inevitable, but Google decided to try to keep the whole system as simple as possible, by designating a single master to be in charge of allocating the chunks in a efficient manner thanks to its global view of the whole system.
- Furthermore, continuing on the last point, reducing the amount of petitions and involvement of this single master is a need, as it could easily become a bottleneck when scaling the whole system. This was managed by designing the master as an address book for the client to ask for which chunkservers to contact, to then do the I/O operations with directly.
- Nevertheless, some design decisions are taken to optimize as much as possible to Google's unique data requirements and settings, which is perfectly valid for their particular uses, but limits the application of the same architecture for other case uses.

Regarding the MapReduce paper:

- One of the main objectives that motivated the development of the MapReduce model was making parallel processing simple enough for programmers

without in depth knowledge and experience in parallel programming and its abstractions capable of creating scripts and applications, without having to worry about all the complexity behind. The resulting programming model accomplishes this without a doubt.

- The paradigm is flexible to be able to express many different problems as MapReduce implementations, which is extremely interesting. Of course, Google developed the whole paradigm to focus on its own computational necessities, but the resulting tool is useful for other many different applications.
- On a personal note, we really enjoyed the MapReduce paper, as it gives many application and implementation examples that show exactly in what type of real-life problems this programming model can be used for.

It is clear that Google has to be on the cutting edge of large-scale distributed architectures, due to their business model requirements and nature. Sadly, one has to admit and realize that the biggest and most relevant advances and breakthroughs in this field are motivated by the needs of the industry, and not by the advancement of the knowledge of academia.