Jumbo: a data intensive distributed computation platform - design overview and preliminary experiment -

Sven Groot
University of Tokyo

Kazuo Goda
University of Tokyo

Masaru Kitsuregawa
University of Tokyo

Abstract

In recent years, the volume of data processed by companies and research institutions has grown enormously, with terabytes and petabytes now being normal. This has led to the development of frameworks for distributed processing of such large quantities of data on large clusters of commodity PCs, such as Google’s MapReduce. However, many of these frameworks sacrifice baseline performance for reliability and scalability. In this paper, we introduce Jumbo, a system designed for experimentation with different approaches on large scale data processing, and outline some of the problems it is intended to solve.

Introduction

Traditional relational database systems do not scale well when processing very large amounts of data on very large clusters. In addition, when dealing with large clusters of commodity PCs, failures are common, and traditional solutions often do not have the required levels of fault tolerance. This means that a failure on a single node during a long-running processing operation would lead to the failure of the job.

To deal with the increasing data processing demands, frameworks have been developed to ease the development of customized distributed data processing solutions.

The most widely known of such frameworks is Google’s MapReduce [1], which provides a programming model based on the map and reduce primitive operations found in many functional programming languages, as well as a fault-tolerant processing environment using Google File System [2] for storage. Hadoop [3] is a well-known open-source implementation of GFS and Map-Reduce.

However, MapReduce is a very strict model and not all processing tasks fit this model, leading to complicated implementations and loss of efficiency. MapReduce also sacrifices efficiency for fault tolerance.

More flexible frameworks have been developed, such as Microsoft Dryad [4], which models jobs as a directed acyclic graph of vertices. While much more flexible, this also makes it more difficult to scale Dryad applications.

We have developed Jumbo, an experimental system designed to evaluate different approaches in data processing, and to investigate alternative design options. The following sections give an overview of Jumbo’s design, and the results of some preliminary experiments.

Design overview

Jumbo consists of two main parts, the Jumbo Distributed File System, and Jumbo Jet, the processing environment. Jumbo DFS is based largely on the design of GFS and Hadoop’s HDFS. A single name server stores file system namespace information, and data servers store file data, divided into blocks of typically 64 or 128MB. Each block is replicated.

Jumbo Jet is the data processing environment for Jumbo, providing a programming model as well as an execution environment.

The programming model divides each data processing job into stages. A stage reads data either from the DFS or from one or more input stages, and writes data either to another stage or to the DFS, forming a directed acyclic graph.

Each stage is divided up into tasks. A task takes part of the input data and performs a processing operation on it. Every task in a stage performs the same operations, just on different parts of the data.

To divide a stage’s input over multiple tasks, it must be split. DFS input is simply split linearly, typically giving each task in the stage a single DFS block as input.

When a stage reads input from another stage, the data from that input stage is partitioned by using a partitioning function. Each task in the stage will receive the same partition from each task in the input stage. This data is not automatically sorted or grouped; the method by which the data of all input tasks is combined can be specified by each stage individually. A stage can choose to have its input data sorted, or simply process it linearly if this is not required, or specify a custom method of combining the data. This offers larger flexibility and significant performance

Jumbo: a data intensive distributed computation platform - design overview and preliminary experiment -
Sven Groot – University of Tokyo
Kazuo Goda – University of Tokyo
Masaru Kitsuregawa – University of Tokyo

benefits when sorting is not required.

Tasks are the basic building blocks of Jumbo Jet jobs. Developers using Jumbo create tasks by writing a function that processes input records and writes output records.

This design offers much greater flexibility than MapReduce. Jumbo can easily simulate MapReduce, but can also use different structures. For example, jobs that would require more than one MapReduce pass can be represented in Jumbo as a single job with multiple stages, eliminating the overhead of creating multiple jobs and storing intermediate data on the DFS. Jumbo's job structure is more rigid than what Dryad allows, which makes it easier to scale and easier for developers to understand the structure of the jobs and debug them.

Jobs are executed in Jumbo using a mechanism similar to Hadoop. A single job server is responsible for scheduling, while task servers, located on the same nodes as the DFS data servers, execute tasks. Intermediate data is stored on disk, so that task or node failures do not fail the entire job.

Experimental results

In order to validate the design of Jumbo, several experiments have been done, comparing performance to Hadoop in multiple instances. Currently, these are still relatively small scale experiments with simple jobs.

One basic job that we have tested is sorting. Hadoop includes a sample implementation for the TeraSort benchmark (now called GraySort). We have created an implementation for the same benchmark in Jumbo. No comparison with Dryad is available because the environment used cannot run Dryad.

To test the performance and scalability, we increased the number of nodes and simultaneously increased the amount of data, keeping the data per node the same, 4GB. Ideally, the processing time for each experiment should be identical.

The results are shown in Figure 1. Jumbo is considerably quicker than Hadoop, over 60% faster on average, despite using a similar sorting methodology. Hadoop's implementation wastes resources by serializing records in-memory and sorting the serialized version, and wastes disk I/O due to the file format used for intermediate files, the merge strategy used, and in some cases also due to speculative execution. Because of the slow disks of the nodes in the cluster used, this waste of disk I/O in particular causes Hadoop to fall short in performance.

Neither quite achieves linear scalability, especially with less than 20 nodes, because merging the data can be done in a single pass in those cases. At 20 nodes, multiple passes are required, and from that point on the scalability improves. We are continuing to reduce communication and other overheads to improve this result.

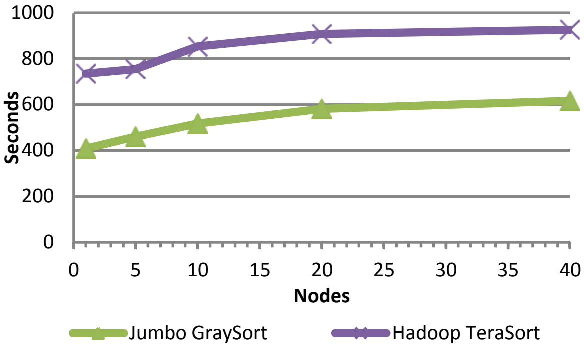

Figure 1 Sorting performance

When using jobs that are not suited to the MapReduce model, even larger differences can be seen. For example, using the word count example from Google's MapReduce paper, Jumbo was up to five times faster than Hadoop.

Conclusion

We have introduced Jumbo, our experimental system for data processing, and shown that it can offer significant performance benefits over Hadoop while maintaining similar levels of scalability and fault tolerance.

There remain many challenges in this field. Our experiments have shown that it is very difficult to tune these systems. With default settings, Hadoop performed considerably worse on the cluster used, and much effort was needed to optimize its performance. We have also seen that these systems are not always able to adequately respond to certain nodes being slower than others, especially in heterogeneous environments.

Our future work is aimed at addressing these issues with Jumbo, as well as using more complicated applications.

Bibliography

1. *MapReduce: Simplified Data Processing on Large Clusters.* **Dean, Jeffrey and Ghemawat, Sanjay.** Berkeley, CA, USA : USENIX Association, 2004. OSDI'04: Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation.

2. *The Google file system.* **Ghemawat, Sanjay, Gobioff, Howard and Leung, Shun-Tak.** New York, NY, USA : ACM Press, 2003. SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles. pp. 29-43.

3. Hadoop Core. *Hadoop.* [Online] Apache. http://hadoop.apache.org/core.

4. *Dryad: distributed data-parallel programs from sequential building blocks.* **Isard, Michael, et al., et al.** New York, NY, USA : ACM, 2007, SIGOPS Oper. Syst. Rev., Vol. 41.