# Fault Tolerance Analysis for Hadoop MapReduce on Gfarm Distributed File System

Graduate School of System and Information engineering
Dept. of Computer Science 1st Year
201120781 Marilia Melo
Supervisor: Osamu Tatebe

## 1 Introduction

In recent years, companies, researches, and goverments accumulate increasingly large amounts of data that they process using advanced analytics. In order to process the data in a reasonable amount of time the computation have to be distributed across hundreds or thousands of computers. As a result, organizations are moving data analysis activities off of the desktop and onto clusters of computers - public and private "clouds". However, programming these clouds for a huge amount of data analysis remains a challenge, since we have to be able to handle machine failures, schedule processes, partition the data and so on. MapReduce[1] has proven itself as a powerful and cost-effective framework for data-intensive computing since it can hide these complexities. It is very beneficial for scientists to use the MapReduce programming model because it makes distributed computation easy.

MapReduce has been used in a variety of applications, including not only log analysis and creating inverted indexes, initially used by Google, but also in genome analytics[2], astronomy[3], and other fields of scientific research. To handle large-scale data, MapReduce utilizes a distributed file system to store the input and output of data. Google File System[4] is used as the input and output of the MapReduce in Google.

Hadoop is a widely used open-source implementation of GFS/MapReduce and has been deployed to multi-thousand node production clusters. Hadoop utilizes the Hadoop Distributed File System (HDFS)[7] to store input and output data. However, HDFS does not support the POSIX semantics since it is not required by MapReduce workloads. It does not support file modification other than the append operation after once closed. Also, HDFS does not support concurrent writes to a single file from multiple clients. Lack of these features makes it difficult for applications other than Hadoop MapReduce to access these file systems, including legacy POSIX applications and MPI-IO applications. We have proposed a solution[10] to this problem by implementing a Hadoop-Gfarm plugin that enables access to the Gfarm file system from Hadoop MapReduce applications. Gfarm file system[9] is a global distributed file system that has a POSIX compliant API and can exploit data locality.

In this research we discuss the use of Hadoo-Gfarm plugin in applications in wide area environment, analysing the use of replication in Hadoop-Gfarm system since Gfarm's replication methods differ greatly from those of HDFS. However, replication is an essential part of modern distributed file systems to provide fault tolerance. We look into providing a solution to applications that lack fault tolerance capabilities when using HDFS, like Impala[8], an open source low latency query engine used for data analytics with Hadoop.

## 2 Background

### 2.1 MapReduce

MapReduce is a popular parallel programming framework for processing large datasets. MapReduce provides a simple API for writing

user-defined operations: a user only needs to specify a serial map function and a serial reduce function. The implementation takes care of applying these functions in parallel to a large set of data. The programming model of MapReduce splits a workflow into 3 phases: map, shuffle and reduce.

$$\begin{aligned} map &:: (K1, V1) \rightarrow [(K2, V2)] \\ reduce &:: (K2, [V2]) \rightarrow [(K3, V3)] \end{aligned}$$

The *map* function takes a key and value of arbitrary types K1 and V1, and returns a sequence of (key, value) pairs of possibly different types, K2 and V2. In the *shuffle* phase, all values associated with the same key K2 are grouped into a sequence and passed to the *reduce* function, which emits arbitrary key-value pairs of a final type K3 and V3.

In the MapReduce architecture there is a single central master node where Job Tracker runs. The job tracker manages all slave/worker nodes and embraces a scheduler that assigns tasks to idle slots. MapReduce master takes the location information of the input files and attempts to schedule a map task on the machine that contains the input file. If that is not possible, it tries to execute on the closest machine to the one holding the input data file. This algorithm conserves network bandwidth and exploits locality to minimize computation time.

#### 2.1.1 Hadoop

Hadoop is open-source implementation of MapReduce currently being developed by Apache Software Foundantion. The Hadoop platform is now commonly considered to consist of the Hadoop kernel, MapReduce and Hadoop Distributed File System (HDFS), as well as a number of related projects - including Apache Hive[6], Apache HBase[5] and others.

### 2.2 Distributed File Systems

This section describes the architecture of HDFS and Gfarm. Although many features are common to both, there are also some important differences.

#### 2.2.1 HDFS

HDFS is a distributed file system, which is normally used by Hadoop. It is designed to hold very large amounts of data and provide high throughput access. Files are split into chunks which are managed by different nodes in the cluster. Each chunk is replicated across several machines, so that a single machine failure does not result in any data being unavailable. However, HDFS relaxes some POSIX requirements in order to achieve high throughput in streaming access.

In scientific research, it is often the case that researchers need to use existing POSIX software such as MATLAB, as well as MPI, which is widely used in high performance computing, cannot run on HDFS. It is often necessary to import files of the POSIX programs to a HDFS, run MapReduce on them, then export the results to a file system that can be read by a POSIX application. Essentially, it needs to execute redundant copy and storage operations.

#### 2.2.2 Gfarm

Gfarm file system is a global distributed file system that is conformable to the POSIX semantics. It has a similar architecture to the HDFS and Google File System in terms of federating local file systems on compute nodes. The Gfarm file system consists of a metadata server (MDS) and I/O servers. The MDS manages the system metadata including a hierarchical namespace, file attributes and the replica catalog. I/O servers provide file access to the local file system. The client can access Gfarm using the Gfarm client library. Also, Linux clients can mount the Gfarm file system using the FUSE kernel module. Files stored in the Gfarm file system can be replicated and stored on any node and accessed by any node.

One big difference between Gfarm and many distributed file systems is that Gfarm does not use file striping or divide the file into blocks, like HDFS. In Gfarm large files are managed by a file group, which is specified by a directory or file name with a wildcard. Using file groups instead of large files gives us one big afvantage over

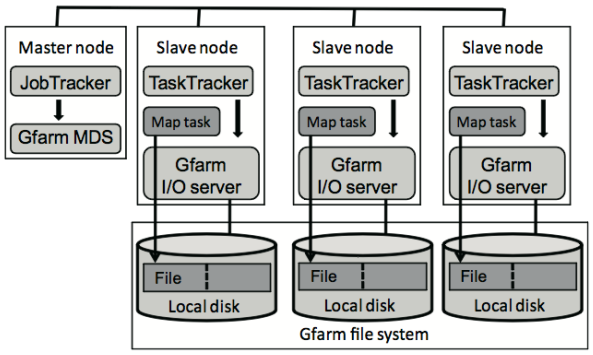


Fig 1: Interaction of Hadoop MapReduce and Gfarm file system

file striping, namely that by splitting large files into groups we can explicitly manage file replica placement. This is key for file location aware distributed data computing.

### 2.3 Hadoop-Gfarm plugin

The physical layout of HDFS and Gfarm are very similar. The NameNode corresponds to the Gfarm MDS, and DataNodes correspond to Gfarm I/O servers. Both file systems federate local file systems to provide a single file system. Therefore, Gfarm can be deployed in the same layout as HDFS. However, HDFS splits files into chunks which are distributed across several machines. Hadoop MapReduce allocates map tasks corresponding these chunks. Meanwhile Gfarm does not split files into chunks, but the disk access pattern of Hadoop MapReduce tasks running on gfarm can be the same as HDFS.

Suppose you run a MapReduce job on HDFS with 4 tasks, each task processes the block pointed to by the indicated line. However, when you run the same MapReduce job on Gfarm, each task processes either the first of last half of its designated files. In this way, MapReduce tasks can be distributed among multiple disks on the Gfarm file system, same as HDFS.

In [10] the Hadoop-Gfarm plugin has been implemented using the Hadoop Common utilities that provides a FileSystem interface to enable access to the Gfarm file system through the Java Native Interface. It contains not only common filesystem APIs such as open, read, write and mkdir, but also getFileBlockLocations to expose the data location of file replicas. Using this interface, Hadoop MapReduce can allocate tasks near input data, as depicted in 1.

## 3 Proposed Fault Tolerance Analysis

Distributed parallel computing requires reliability to be useful. A single computer may fail once a year. With 365 computers, one will fail everyday. If you have a cluster with 36,500 computers, failures will occur every hour. Reliability is an essential feature for cloud computing processings. Hadoop-Gfarm plugin has been implemented without use of replication, making it not a reliable system.

In this research we aim to provide fault tolerance to Hadoop-Gfarm plugin by making use of replication. Different than HDFS, the Gfarm allows to replicate not only the datanodes with processing data, but also the Master node and its metadata information. This solves the Single Point of Failure that exists on HDFS. To implement such feature we need to install Hadoop, Gfarm and then Hadoop-Gfarm plugin to be able to edit the replication configuration. Gfarm provides the *gfrep* command to create replicas but also has automatic replication system. We explain the basic replication system available in Gfarm in the next session.

### 3.1 Replication in Distributed File Systems

Replication methods are available in both Hadoop and Gfarm, but differ greatly. HDFS uses chain replication, meaning that by the time the client finishes writing the data, the replicas have already been created.

However, Gfarm create replicas in the background after the client has finished writing the data. Even if you create replicas, write performances should not change.

## 4 Related Work

Trying to use a different distributed file system with Hadoop has been explored in many ocasions, but most of them requires changes to the system configuration or does not support most

important features. In [11] the authors compare HDFS and PVFS, and show that PVFS can perform as well as HDFS. However, they have changed the configuration, increasing the default 64 KB stripe size to 64 MB, the same as the HDFS block size. This change may cause performance degradation in other applications. CloudStore [12] is a distributed file system integrated with Hadoop through the Java Native Interface. However, CloudStore lacks many of the required features necessary for a general purpose file system, such as security features and file permissions. The GPFS on Hadoop Project [13] allows Hadoop MapReduce applications to access files on GPFS. However, it requires changing block size to 128 MB for MapReduce and 128 KB block size for online transaction processing. This means the data cannot really be shared by MapReduce applications and other applications even in the same file system since the optimal block size is different.

## 5 Conclusion and Work Proposal

In this paper we analyzed the currently mainly used solutions for High Performance Cloud Computing as a file system to the MapReduce framework. The MapReduce framework has been more and more utilized for data analysis processing of large scale data. Distributed File System with POSIX compliance is still a strong requirement for scientific applications. The Hadoop-Gfarm plugin enables Hadoop MapReduce to be excute on Gfarm File System. However, this plugin does not provide fault tolerance capabilities, which is essential in a scalable distributed computing system.

As the next steps for our work, we plan to deploy th Hadoop-Gfarm plugin using replication to provide fault tolerance. We will then benchmark the performance of the new Hadoop-Gfarm plugin with this new reliability layer. Since Gfarm's replication structure runs in the background, the replication should not affect performance results. Finally we plan to experiment the system with real time applications, like Impala[8]. This query engine currently runs over HDFS but does not present any reliability, limitating its use in real life systems.

## References


[1] Jeffrey Dean, Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". Proceedings of OSDI '04, 2004.

[2] Ben Langmead, Michael C Schatz, Jimmy Lin, Mihai Pop and Steven L Salzberg, "Searching for snps with cloud computing," Genome Biol 2009, 10:R134, 2009.

[3] Ewa Deelman, Gurmeet Singh, Miron Livny, Bruce Berriman, John Good, "The cost of doing science on the cloud: The montage example", 2008.

[4] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google le system", *SIGOPS - Operating Systems Review*, Vol. 37, No. 5, pp. 29—43, 2003.

[5] Apache, "HBase". `http://wiki.apache.org/hadoop/hbase`.

[6] Apache, "Hive". `http://wiki.apache.org/hadoop/hive`.

[7] Apache,"HDFS architecture". `http://hadoop.apache.org/common/docs/current/hdfsdesign.html`.

[8] Cloudera Enterprise RTQ, `http://www.cloudera.com/content/cloudera/en/products/cloudera-enterprise-core/cloudera-enterprise-RTQ.html`.

[9] Osamu Tatebe, Kohei Hiraga, Noriyuki Soda, "Gfarm grid le system", in New Generation Computing, Ohmsha, Ltd. and Springer, Vol.28, No.3, pp.257-275, DOI: 10.1007/s00354-009-0089-5, 2010.

[10] Kazuki Ohta and Shunsuke Mikami. Hadoop-Gfarm. `https://gfarm.svn.sourceforge.net/svnroot/gfarm/gfarm_hadoop/trunk/`.

[11] Wittawat Tantisiriroj, Swapnil Patil, and Garth Gibson, "Data-intensive file systems for internet services: A rose by any other", *CMU-PDL-08- 114*, 2008.

[12] "CloudStore," in `http://kosmosfs.sourceforge.net`.

[13] Karan Gupta, Reshu Jain, Himabindu Pucha, Prasenjit Sarkar, Dinesh Subhraveti, "Scaling highly-parallel data-intensive supercomputing applications on a parallel clustered le system," *The SC10 Storage Challenge*, 2010.