Vol.2013-HPC-140 No.14
2013/7/31

Fault Tolerance Design for Hadoop MapReduce on Gfarm Distributed Filesystem

Marilia Melo[1,a)] Osamu Tatebe[1,b)]

Abstract: Many distributed file systems that have been designed to use MapReduce, such as Google file system and HDFS (Hadoop Distributed File System), relax some POSIX requirements to enable high throughput streaming access. Due to lack of POSIX compatibility, it is difficult for programs other than MapReduce to access these file systems. It is often need to import files to these file system, process the data and then export the output into a POSIX compatible file system. This results in a large number of redundant file operations. In order to solve this problem we have proposed[9] Hadoop-Gfarm plugin to be able to execute MapReduce jobs directly on top of Gfarm, a globally distributed file system. In this paper we analyse the redundancy and reliability for a fault tolerance design for Hadoop-Gfarm plugin. Our evaluation shows that Hadoop-Gfarm plugin can offer a reliable solution and performs just as well as Hadoop's native HDFS, allowing users to use a POSIX-compliant API and reduces redundant copy without sacrificing performance.

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[9] to this problem by implementing a Hadoop-Gfarm plugin that enables access to the Gfarm file system from Hadoop MapReduce applications. Gfarm file system[8] 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, and analyse the use of replication in Hadoop-Gfarm to provide fault tolerance capability, 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 reliable solution to applications that make use of POSIX-compliante systems without sacrificing performance by using Hadoop-Gfarm plugin with replication.

This paper is structured as follows. Section 2 introduces MapReduce and Hadoop technologies. Section 3 describes the Gfarm file system and Gfarm-Hadoop plugin and Section 4 describes fault tolerance by analysing the replication process. Section 5 shows the performance evaluation. Section 6 introduces related work and section 7 concludes this paper.

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

[1] Graduate School of Systems and Information Engineering, University of Tsukuba
[a)] marilia@hpcs.cs.tsukuba.ac.jp
[b)] tatebe@cs.tsukuba.ac.jp

set of data. The programming model of MapReduce splits a workflow into 3 phases: map, shuffle and reduce.

$$map :: (K1, V1) \rightarrow [(K2, V2)]$$
$$reduce :: (K2, [V2]) \rightarrow [(K3, V3)]$$

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 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.

3. Gfarm and Gfarm-Hadoop plugin

3.1 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.

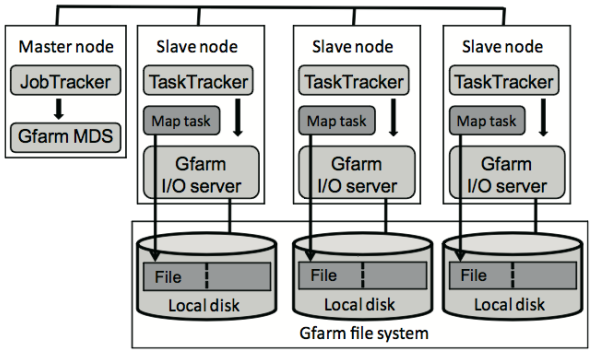

Fig. 1 Interaction of Hadoop MapReduce and Gfarm file system

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 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.

3.2 Gfarm-Hadoop 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 [9] 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.

4. Fault Tolerance by Replication

Fault-tolerance is the property that enables a system to continue operating properly in the event of the failure of one more

components and not lose data even after some components of that system have failed. In managing fault tolerance it is important to eliminate Single Points of Failure (SPOF) of a system. The current stable version of Hadoop, 1.1 release, provides a high degree of fault tolerance to your jobs by restarting tasks and running speculative executions. However, on HDFS the master node (NameNode) is still a SPOF and if it goes down, the system is unavailable.

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, i.e. providing multiple identical instances of the same system or subsystem, directing tasks or requests to all of them in parallel. 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 can solve 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* and *gfncopy* commands to create replicas but also has automatic replication system. We explain the basic replication system available in HDFS and Gfarm in the next session.

4.1 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.

4.1.1 Data Replication in HDFS

When writing data to a hadoop cluster, the client breaks the data file into ”Blocks” and place those blocks on different machines throughout the cluster. Each block will be replicated in the cluster as its loaded. The standard setting for Hadoop is to have 3 copies of each block in the cluster. This can be configured with the *dfs.replication* parameter in the *hdfs-site.xml*.

As data for each block is written into the cluster a replication pipeline is created between the data nodes, meaning that as a data node is receiving block data it will at the same time push a copy of that data to the next node in the pipeline. We can easily understand that Hadoop uses a lot of network bandwidth and storage. Since Hadoop usually treats very big files and each file will be replicated onto the network and disk 3 times.

The current gfarm-hadoop plugin was implemented and tested with dfs.replication parameter set to 1, that means that only one version of the data file was stored and replicas were not created in the system.

4.1.2 Data Replication in Gfarm

Gfarm has many ways to support replication[14]. The biggest difference from HDFS and Gfarm replication is that the current

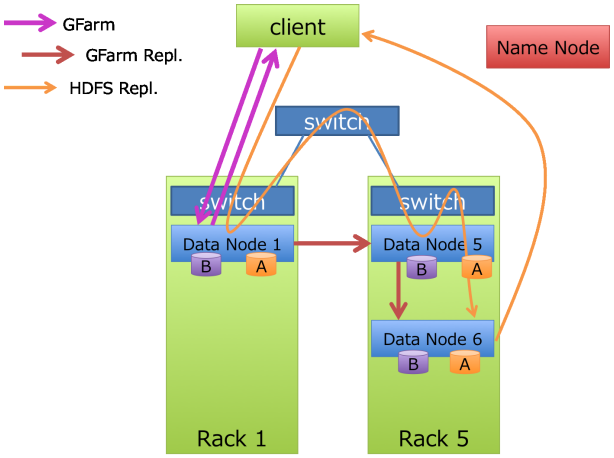

Fig. 2 Replica system on HDFS and Gfarm

version of Gfarm replication process is in asynchronous mode. That means the replication only happens when a file is closed by its last writer. For a MapReduce application, it means the network bandwidth and system CPU will not be affected during the execution of the job.

To create replicas on Gfarm you can use the *gfrep*, that will generate the specified number of copies on the spot, and you can also use the *gfncopy*, that manipulate the number of replicas automatically created for a filesystem. The later command adds an extended attribute to the specified directory that every time a new file is created under that directory, automatically number of replicas will also be created.

In the same way as HDFS checks Datanodes availability periodically, Gfarm also make sure all replicas are available for the user and when a node goes down, it will automatically replicate the data in another node.

The figure 2 highlights the difference of replication system between HDFS and Gfarm.

5. Performance Evaluation

5.1 Evaluation environment and configuration tunning

Performance evaluation for Gfarm-hadoop plugin regarding scaling with number of cluster nodes has been already demonstrated by [13]. So in this paper we focused on analysing the implementation example of a running cluster available on InTrigger system[15] with 14 nodes, with 1 machine dedicated to Metadata Server/Name Node and Job Tracker, and 13 compute nodes. Each machine has the configuration as described in **Table 1** and the software configuration as listed in **Table 2**.

Hadoop has an extensive set of configurations available for tunning its performance. This time we made the minimum required changes from the default values. The used settings are listed in

Table 1 Machine Specification

CPU	2.4GHz Quadcore Xeon E5620 (2 sockets)
Memory	24GB
Disk	ST9500430SS 500GB

Table 2 Software Specification

OS	Linux 2.6.26-2-amd64 SMP
Hadoop	1.1.2
Gfarm	2.5.8.1

Table 3 Hadoop Settings

Property Name	Property Value
mapred.tasktracker.map.tasks.maximum	2
mapred.tasktracker.reduce.tasks.maximum	2
mapred.map.tasks	26
mapred.reduce.tasks	13
dfs.replication	3
dfs.block.size	134217728

Table 3.

Also, to make sure Gfarm had the replication settings turned on automatically, a path for /home/$USER was created with the *gncopy* attribute set to 3. This way, everytime a file is written under /home/$USER on gfarm, 3 replicas will be automatically created in different nodes. You can check the location of the replicas by using the command *gfwhere*.

5.2 Benchmarks

Hadoop provides pre-installed benchmarks like TeraGen, TeraSort and TeraValidate. We used a mixed of benchmarks for this analysis, executing 4 kinds of benchmarks to analyze the write, read, sort and grep functions.

We chose the TeraSort benchmark because it is probably the most well-known Hadoop benchmark, since in 2009 Yahoo! set a record by sorting 1 PB of data in 16 hours in a Hadoop cluster of 3800 nodes[16]. In order to test the write performance, we executed TeraGen benchmark to generate 10GB of data. TeraGen generates random data that can be used as input for a subsequent TeraSort run. It uses only Map tasks and is used to evaluate the write performance of the node.

For TeraSort applications the data size of the input and output are the same. It has a initial Map phase and also Reduce phase, so this bechmark is used to test the write and read capabilities.

TeraValidate creates one map task per file in TeraSort's output directory. The map task ensures that each key is less than or equal to the previous one and also generates records with the first and last keys of the file. The reduce tasks ensures that the first key of file *i* is greater than key of file *i*-1. This benchmark tests the read function.

Finally we used the grep application. It scans through all the data, searching for the given input string. Grep benchmark is a read-intensive application because the output data size if much smaller than the input data size.

However, since Gfarm replication takes place in the backend, we executed these 4 benchmarks in sequence in one line only command. By doing so, we want to make sure the next process can run while the replication is being executed in the backend. Each set of the 4 benchmarks was executed 5 times and we used the average the represent the time execution for each benchmark individually.

5.3 Results

The results from this experiment are shown on 3. As we can see, the Teragen benchmark is 57% faster on Gfarm than HDFS. We believe this is due to the fact HDFS finishes writing all its replicas before closing the file, as we explained on the previous section.

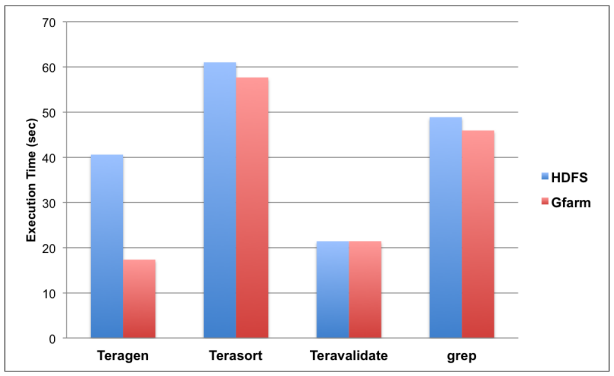

Fig. 3 Benchmark results for Gfarm and HDFS with replica

For the following benchmarks there are not so significant differences. Gfarm is sligthly faster than HDFS on average, but this is less than 5%. We can understand that for initial mapreduce jobs, the Gfarm system performes higher than HDFS. This is probably because Gfarm usually writes to the filesystem once and finishes the job, while HDFS writes all the 3 replicas at the same time. Also, during the first job there is no replication process being executed on the backend, so the overhead is relatively low when compared to the following jobs. However, for the following jobs, the performance does not change significantly. This may be showing that backend replicas does not affect the normal MapReduce processes on Gfarm, but does not add performance improvent when compared to HDFS.

This results shows that using Hadoop with Gfarm with replicas is a realiable way to use a POSIX-compliant filesystem with equal performance as HDFS. We still need to averiguate the real cases where the user needs to transfer the data input to the distributed filesystem. Gfarm can provide a solution to this transfer period, while HDFS requires double transfer for import and export of the data. We also want to verify the amount of real jobs that can be executed with single mapreduce jobs, and those that need to run many interactions.

6. 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 [10] 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 [11] 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 [12] 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.

Also, recent projects have tried to make hadoop more reliable with projects called Hadoop High Availability[17]. The next version of Apache Hadoop 2.0.5, still in alpha version, includes a new concept of Standby NameNode, a secondary namenode that would assume control in case the main namenode fails. However, this is still in development state.

7. Conclusion

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.

The HDFS filesystem creates replicas at the moment of writing data to the system. If the replica settings is set to 3, the system will write the data to 3 locations at the same time before finalizing the process. However, the Gfarm filesystem has an assynchronous way of creating replicas. It means that the data is initial written to one node and after the client finalizes the process, the replication then takes place.

We have implemented a fault tolerance design to gfarm-hadoop plugin by making use of replicas, that can be configured to be automatically created. Using of the default benchmarks provided by Hadoop, we tested the system with the Teragen, Terasort and Teravalidate benchmarks. Since Gfarm's replication structure runs in the background, the replication does not affect performance results. More over, for the initial mapreduce jobs, the Gfarm application can be 57% faster than HDFS, since HDFS finishes writing all its replicas before closing the file.

Our future work will include performance evaluation using systems like Pig[18], a platform for analyzing large data sets that consists of a compiler that produces sequences of Map-Reduce programs. By having such benchmarks where mapreduce tasks are created sequentially, we could analyze in detail with the background replication influences the system performance or not. Also, we want to analyze examples where you need to import your data. Since HDFS is not POSIX-compliant, most of the applications need to spend time transfering the data in and out of the filesystem. Gfarm-hadoop allows the user to save this time and not sacrifice performance.

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 file system", *SIGOPS - Operating Systems Review*, Vol. 37, No. 5, pp. 2943, 2003.
[5] Apache, "HBase". DOI: http://wiki.apache.org/hadoop/hbase.
[6] Apache, "Hive". DOI: http://wiki.apache.org/hadoop/hive.
[7] Apache,"Hadoop Distributed File System architecture". DOI: http://hadoop.apache.org/common/docs/current/hdfs design.html.
[8] Osamu Tatebe, Kohei Hiraga, Noriyuki Soda, "Gfarm grid file system", in New Generation Computing, Ohmsha, Ltd. and Springer, Vol.28, No.3, pp.257-275, DOI: 10.1007/s00354-009-0089-5, 2010.
[9] Kazuki Ohta and Shunsuke Mikami. Hadoop-Gfarm. DOI: https://gfarm.svn.sourceforge.net/svnroot/gfarm/gfarm_hadoop/trunk/.
[10] Wittawat Tantisiriroj, Swapnil Patil, and Garth Gibson, "Data-intensive file systems for internet services: A rose by any other", *CMU-PDL-08- 114*, 2008.
[11] "CloudStore," in DOI: http://kosmosfs.sourceforge.net.
[12] Karan Gupta, Reshu Jain, Himabindu Pucha, Prasenjit Sarkar, Dinesh Subhraveti, "Scaling highly-parallel data-intensive supercomputing applications on a parallel clustered file system," *The SC10 Storage Challenge*, 2010.
[13] Shunsuke Mikami, Kazuki Ohta, Osamu Tatebe: "Using the Gfarm File System as a POSIX compatible storage platform for Hadoop MapReduce applications", IEEE/ACM International Conference on Grid Computing (Grid 2011).
[14] Grid Datafarm, DOI: http://datafarm.apgrid.org/.
[15] InTrigger, DOI: http://www.intrigger.jp/.
[16] Hadoop Sorts a Petabyte in 16.25 Hours and a Terabyte in 62 Seconds, DOI: http://developer.yahoo.com/blogs/hadoop/hadoop-sorts-petabyte-16-25-hours-terabyte-62-422.html.
[17] HDFS High Availability, DOI: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/HDFSHighAvailabilityWithNFS.html.
[18] Apache Pig, DOI: http://pig.apache.org/.