Applied Mathematics & Information Sciences
An International Journal

Software Design and Implementation for MapReduce across Distributed Data Centers

Lizhe Wang[1,5,*], *Jie Tao*[2], *Yan Ma*[1], *Samee U. Khan*[3], *Joanna Kołodziej*[4], *Dan Chen*[5,*]

[1] Center for Earth Observation and Digital Earth, Chinese Academy of Sciences, P. R. China
[2] Steinbuch Center for Computing, Karlsruhe Institute of Technology, Germany
[3] Department of Electrical and Computer Engineering, North Dakota State University, USA
[4] Institute of Computer Science, Cracow University of Technology, Poland
[5] School of Computer Science, China University of Geosciences, P. R. China

Received: 29 Aug. 2012, Revised: 24 Nov. 2012, Accepted: 4 Dec. 2012
Published online: 1 Feb. 2013

Abstract: Recently, the computational requirements for large-scale data-intensive analysis of scientific data have grown significantly. In High Energy Physics (HEP) for example, the Large Hadron Collider (LHC) produced 13 petabytes of data in 2010. This huge amount of data are processed on more than 140 computing centers distributed across 34 countries. The MapReduce paradigm has emerged as a highly successful programming model for large-scale data-intensive computing applications. However, current MapReduce implementations are developed to operate on single cluster environments and cannot be leveraged for large-scale distributed data processing across multiple clusters. On the other hand, workflow systems are used for distributed data processing across data centers. It has been reported that the workflow paradigm has some limitations for distributed data processing, such as reliability and efficiency. In this paper, we present the design and implementation of G-Hadoop, a MapReduce framework that aims to enable large-scale distributed computing across multiple clusters. G-Hadoop uses the Gfarm file system as an underlying file system and executes MapReduce tasks across distributed clusters.

Keywords: Cloud computing, MapReduce, Data-intensive computing, data center computing

1. Introduction

The rapid growth of Internet and WWW has led to vast amounts of information available online. In addition, social, scientific and engineering applications have created large amounts of both structured and unstructured information which needs to be processed, analyzed, and linked [1,2]. Nowadays data-intensive computing typically uses modern data center architectures and massive data processing paradigms. This research is devoted to a study on the massive data processing model across multiple data centers.

The requirements for data-intensive analysis of scientific data across distributed clusters or data centers have grown significantly in the recent years. A good example for data-intensive analysis is the field of High Energy Physics (HEP). The four main detectors including ALICE, ATLAS, CMS and LHCb at the Large Hadron Collider (LHC) produced about 13 petabyes of data in 2010 [3,4]. This huge amount of data are stored on the Worldwide LHC Computing Grid that consists of more than 140 computing centers distributed across 34 countries. The central node of the Grid for data storage and first pass reconstruction, referred to as Tier 0, is housed at CERN. Starting from this Tier, a second copy of the data is distributed to 11 Tier 1 sites for storage, further reconstruction and scheduled analysis. Simulations and USER ANALYSIS are performed at about 140 Tier 2 sites. In order to run the latter, researchers are often forced to copy data from multiple sites to the computing centre where the DATA ANALYSIS is supposed to be run. Since the globally distributed computing centers are interconnected through wide-area networks the copy process is tedious and inefficient. We believe that moving the computation instead of moving the data is the key to tackle this problem. By using data parallel processing paradigms on multiple clusters, simulations can be run on multiple computing centers concurrently without the need of copying the data.

* Corresponding author e-mail: Lizhe.Wang@gmail.com

Currently data-intensive workflow systems, such as DAGMan [5], Pegasus [6], Swift [7], Kepler [8], Virtual Workflow [9, 10], Virtual Data System [11] and Taverna [12], are used for distributed data processing across multiple data centers. There are some limitations for using workflow paradigms across multiple data centers: 1) Workflow system provides a coarse-grained parallelism and cannot fulfill the requirement of high throughput data processing, which typically demands a massively parallel processing. 2) Workflow systems for data intensive computing typically requires large data transfer for between tasks, sometime it brings unnecessary data blocks or data sets movement. 3) Workflow systems have to take care of fault tolerance for task execution and data transfer, which is not a trivial implementation for data intensive computing. Given the wide acceptance of the MapReduce paradigm, it would be natural to use MapReduce for data processing across distributed data centers, which can overcome the aforementioned limitations of workflow systems.

In this paper, we present the design and implementation of G-Hadoop, a MapReduce framework that aims to enable large-scale distributed computing across multiple clusters. In order to share data sets across multiple administrative domains, G-Hadoop replaces the Hadoop's native distributed file system with the Gfarm file system. Users can submit their MapReduce applications to G-Hadoop, which executes map and reduce tasks across multiple clusters.

G-Hadoop provides a parallel processing environment for massive data sets across distributed clusters with the widely-accepted MapReduce paradigm. Compared with data-intensive workflow systems, it implements a fine-grained data processing parallelism and achieves high throughput data processing performance. Furthermore, by duplicating map and reduce tasks G-Hadoop can provide fault tolerance for large-scale massive data processing.

The rest of this paper is organized as follows: Section 2 discusses background and related work of our research; Section 3 and Section 4 present the design and implementation of G-Hadoop. Finally Section 5 concludes the paper and points out the future work.

2. Background and related work

A computing Cloud is a set of network enabled services, providing scalable, QoS guaranteed, normally personalized, inexpensive computing infrastructures on demand, which can be accessed in a simple and pervasive way [13]. Conceptually, users acquire computing platforms, or IT infrastructures from computing Clouds and execute their applications. Therefore, computing Clouds render users with services to access hardware, software and data resources, thereafter an integrated computing platform as a service.

The MapReduce [14] programming model is based on two main procedures in functional programming: Map and Reduce. The Map function processes key/value pairs to generate a set of intermediate key/value pairs and the Reduce function merges all the same intermediate values. Many real-world applications are expressed using this model.

The Apache Hadoop project [15], the mostly used MapReduce implementation, develops open-source software for reliable, scalable massive data processing with the MapReduce model. It contains 1) Hadoop Distributed File System (HDFS), a distributed file system that provides high-throughput access to application data, and 2) Hadoop MapReduce, a software framework for distributed processing of large data sets on compute clusters.

The Gfarm file system [16] is a distributed file system designed to share vast amounts of data between globally distributed clusters connected via a wide-area network. Similar to HDFS the Gfarm file system leverages the local storage capacity available on compute nodes. A dedicated storage cluster (SAN) is not required to run the Gfarm file system.

The main task of a Distributed Resource Management System (DRMS) for a cluster is to provide the functionality to start, monitor and manage jobs. In our initial implementation, we use the Torque Resource Manager [17] as a cluster DRMS. Distributed Resource Management Application API (DRMAA) [18] is a high-level API specification for the submission and control of jobs to one or more DRMSs within a distributed Grid architecture. In this research, we use DRMAA as an interface for submitting tasks from G-Hadoop to the Torque Resource Manager.

3. System Design of G-Hadoop

3.1. Target environment and development goals

Our target environments for G-Hadoop are multiple distributed High End Computing (HEC) clusters. These clusters typically consist of specialized hardware interconnected with high performance networks such as Infiniband. The storage layer is often backed by a parallel distributed file system connected to a Storage Area Network (SAN). HEC clusters also typically employ cluster scheduler, such as Torque, in order to schedule distributed computations among hundreds of compute nodes. Users in HEC clusters generally submit their jobs to a queue managed by the cluster scheduler. When the requested number of machines becomes available, jobs are dequeued and launched on the available compute nodes.

We keep the following goals when developing G-Hadoop:

–Minimal intrusion: When leveraging established HEC clusters with G-Hadoop, we try to keep the autonomy of the clusters, for example, insert software modules in the cluster head node and only execute tasks by talking with a cluster scheduler.
–Compatibility: The system should keep the Hadoop API and be able to run existing Hadoop MapReduce programs without or only with minor modifications of the programs.

3.2. Architecture overview

The proposed architecture of G-Hadoop represents a master/slave communication model. Figure 1 shows an overview of the G-Hadoop's high-level architecture and its basic components: the G-Hadoop Master node and the G-Hadoop Slave nodes.

For simplicity of illustration assume that the G-Hadoop Master node consolidates all software components that are required to be installed at a central organization that provides access to the G-Hadoop framework. A G-Hadoop Slave node, on the other hand, consolidates all software components that are supposed to be deployed on each participating cluster. However, there is no such requirement in our design, each software component can be installed on individual nodes for reasons of performance or availability.

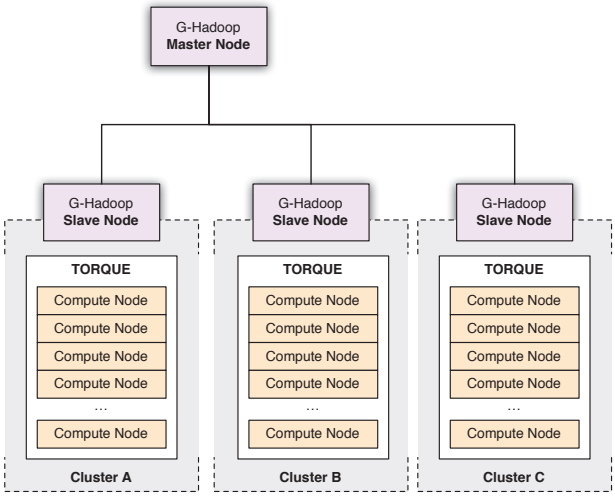


Figure 1 Architecture Overview of G-Hadoop

3.3. Gfarm as a global distributed file system

The MapReduce framework for data-intensive applications heavily relies on the underlying distributed file system. In traditional Hadoop clusters with HDFS, map tasks are preferably assigned to nodes where the required input data is locally present. By replicating the data of popular files to multiple nodes, HDFS is able to boost the performance of MapReduce applications.

In G-Hadoop we aim to schedule MapReduce applications across multiple data centers interconnected through wide-area networks. Hence, applications running concurrently on different clusters must be able to access the required input files independent of the cluster they are executed on. Furthermore, files must be managed in a site-aware manner in order to provide the required location information for the data-aware scheduling policy on the *JobTracker*.

G-Hadoop uses the Gfarm file system as its underlying distributed file system. The Gfarm file system was specifically designed to meet the requirements of providing a global virtual file system across multiple administrative domains. It is optimized for wide-area operation and offers the required location awareness to allow data-aware scheduling among clusters.

3.4. G-Hadoop Master Node

The master node is the central entity in the G-Hadoop architecture. It is responsible for accepting jobs submitted by the user, splitting the jobs into smaller tasks and distributing these tasks among its slave nodes. The master is also responsible for managing the metadata of all files available in the system. The G-Hadoop master node depicted in Figure 2 is composed of the following software components:

–Metadata Server: This server is an unmodified instance of the Metadata server of the Gfarm file system. The metadata server manages files that are distributed among multiple clusters. It resolves files to their actual location, manages their replication and is responsible for taking track of opened file handles in order to coordinate access of multiple clients to files. The Gfarm metadata server is also responsible for managing users access control information.
–*JobTracker*: This server is a modified version of Hadoop's original *JobTracker*. The *JobTracker* is responsible for splitting jobs into smaller tasks and scheduling these tasks among the participating clusters for execution. The *JobTracker* uses a data-aware scheduler and tries to distribute the computation among the clusters by taking the data's locality into account. The Gfarm file system is configured as the default file system for the MapReduce framework. The Gfarm Hadoop plug-in acts as glue between Hadoop's MapReduce framework and the Gfarm file system.

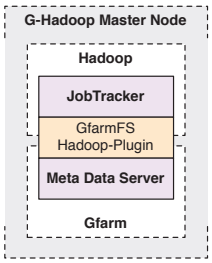


Figure 2 Software components of the G-Hadoop master node

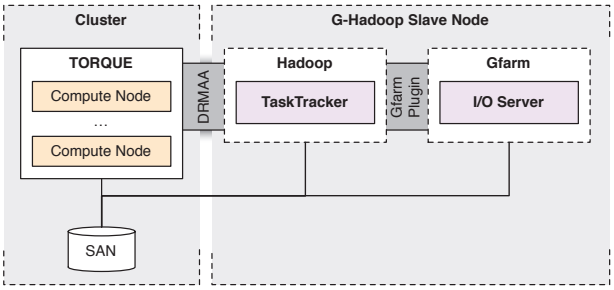


Figure 3 Software components of G-Hadoop's slave node

3.5. G-Hadoop Slave Node

A G-Hadoop slave node is installed on each participating cluster and enables it to run tasks scheduled by the *JobTracker* on the G-Hadoop master node. The G-Hadoop slave node (see Figure 3) consists of the following software components:

- *TaskTracker*: This server is an adopted version of Hadoop *TaskTracker* and includes G-Hadoop related modifications. The *TaskTracker* is responsible for accepting and executing tasks sent by the DRMAA Gfarm Plugin.
- *JobTracker*. Tasks are submitted to the queue of the cluster scheduler (e.g. Torque) using a standard DRMAA interface. A DRMAA java library is used by the *TaskTracker* for task submission. Depending on the distributed resource manager used in the corresponding cluster, an adopted library is required. In order to access the files stored on the Gfarm file system, the Gfarm Hadoop plug-in is used.
- I/O Server: A Gfarm I/O server that manages the data stored on the G-Hadoop slave node. The I/O server is paired with the Metadata server on the G-Hadoop master node and is configured to store its data on a the high performance file system on the cluster. In order to address performance bottlenecks, additional nodes with I/O servers can be deployed on the individual clusters.
- Network Share: The MapReduce applications and their configuration are localized by the *TaskTracker* to a shared location on the network. All compute nodes of the cluster are required to be able to access this shared location with the localized job in order to be able to perform the jobs execution. In addition, the network share is used by the running map tasks on the compute nodes to store their intermediate output data. Since this data is served by the *TaskTracker* to a reduce task the performance of the network share is crucial and depends highly on the performance of the underlying network.

4. Implementation of G-Hadoop

4.1. TaskController of Hadoop

Before describing the implementation of G-Hadoop, it is useful to rapidly recall some basic concepts of tasks and their execution in Hadoop.

In Hadoop, the *JobTracker* is responsible for splitting scheduled MapReduce jobs into smaller entities called tasks. The tasks are then distributed among multiple *TaskTracker*s for execution. Each *TaskTracker* is configured to concurrently execute a certain amount of tasks, referred to as slots. *TaskTracker*s periodically report the current status of each slot to the *JobTracker*. If a *TaskTracker* reports an idle slot, the *JobTracker* usually responds with a new task enclosed in the heartbeat response. When the *TaskTracker* receives a new task for execution, it spawns a new JVM that is configured to run the task. Hadoop provides a new abstract class *TaskController* for managing the execution of tasks. Each *TaskTracker* employs a single instance of the *TaskConroller*. The concrete implementation used can be configured for each *TaskTracker* individually by setting the parameter *mapreduce.tasktracker.taskcontroller* in the *mapred-site.xml* configuration file. If no explicit configuration is provided, an instance of the class *TaskController* is used.

4.2. DRMAA bindings for Torque

The G-Hadoop's architecture uses the standard DRMAA interface for submitting tasks to the cluster scheduler. Libraries for Java can be obtained from GridWay and include the abstract interfaces that can be freely used by application programmers. There are three main Java interfaces provided by GridWay. However, the GridWay library requires custom implementations in order to be able to communicate with the desired cluster scheduler. Using the configuration option *–enable-drmaa* Torque can be configured to compile a native DRMAA 1.0 library that can be employed by application programmers for submitting jobs to Torque. In order to provide

implementations of the GridWay interfaces for Torque in Java, bindings to the native DRMAA library are required. Inspired by bindings for the Sun Grid Engine we implement custom bindings to the Torque using the Java Native Interface (JNI). In G-Hadoop each *TaskTracker* can be configured to use its own implementation of the GridWay DRMAA interfaces by setting the parameter *org.ggf.drmaa.SessionFactory* in the *mapred-site.xml* configuration file. This allows G-Hadoop to use any DRMAA compatible cluster scheduler without the requirement to change the source code base of G-Hadoop.

4.3. DRMAA task controller

Our prototype uses a custom implementation of the *TaskController* interface in order to submit tasks to a cluster scheduler using the DRMAA library described in the previous section. This section presents a detailed description of the class *TaskController* and some of the most relevant implementation details of the DRMAA *TaskController*. The *TaskController* is used by the *TaskTracker* in order to setup, start and stop tasks that are scheduled for execution on the *TaskTracker*.

4.4. Adoptions on JobTracker

When the *JobTracker* receives a heartbeat message from an idle *TaskTracker*, it looks for queued map and reduce tasks and assigns them to the *TaskTracker* for execution. It is essential to the G-Hadoop architecture that multiple tasks are assigned in one heartbeat response. In G-Hadoop a single *TaskTracker* is supposed to execute hundreds of tasks in parallel. Assignment of a single task per heartbeat would dramatically impair the performance of the system.

Hadoop's current implementation of the *JobTracker* is able to assign multiple map tasks to a single *TaskTracker* in one heartbeat response. However, it limits the number of reduce tasks per *TaskTracker* to one. We implemented a new configuration option that allows the *TaskTracker* to be configured not to limit the number of concurrent reduce tasks per *TaskTracker*.

5. Conclusion

The goal of this research is to advance the MapReduce framework for large-scale distributed computing across multiple data centers with multiple clusters. The framework supports distributed data-intensive computation among multiple administrative domains using existing unmodified MapReduce applications. In this work, we have presented the design and implementation of G-Hadoop, a MapReduce framework based on Hadoop that aims to enable large-scale distributed computing across multiple clusters. The architecture of G-Hadoop is based on a master/slave communication model. In order to support globally distributed data-intensive computation among multiple administrative domains, we use the traditional HDFS file system with the Gfarm file system, which can manage huge data sets across distributed clusters.

We have managed to keep the required changes on existing clusters at a minimum in order to foster the adoption of the G-Hadoop framework. Existing clusters can be added to the G-Hadoop framework with only minor modifications by deploying a G-Hadoop slave node on the new cluster. The operation of the existing cluster scheduler is not affected in our implementation. Our work is fully compatible with the Hadoop API and does not require modification of existing MapReduce applications.

Finally we validated our design by implementing a prototype based on the G-Hadoop architecture. It executes MapReduce tasks on the Torque cluster scheduler.

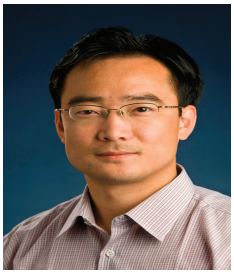
Acknowledgement

Dr. Lizhe Wang's work in this book was partially supported by "One Hundred Talents Programme" of Chinese Academy of Sciences.
Dr. Dan Chen is funded by National Natural Science Foundation of China (grant No.61272314), the Hundred University Talent of Creative Research Excellence Programme (Hebei, China), the Fundamental Research Funds for the Central Universities (CUG, Wuhan), the Specialized Research Fund for the Doctoral Program of Higher Education (grant No. 20110145110010), the Excellent Youth Foundation of Hubei Scientific Committee (grant No. 2012FFA025), and the Program for New Century Excellent Talents in University (grant No. NCET-11-0722).

References

[1] F. Berman, "Got data?: a guide to data preservation in the information age," *Commun. ACM*, vol. 51, pp. 50–56, December 2008.
[2] L. Wang, M. Kunze, J. Tao, and G. von Laszewski, "Towards building a cloud for scientific applications," *Advances in Engineering Software*, vol. 42, no. 9, pp. 714–722, 2011.
[3] G. Brumfiel, "High-energy physics: Down the petabyte highway," *Nature*, no. 7330, pp. 282–283, January 2011.
[4] L. Wang and C. Fu, "Research advances in modern cyberinfrastructure," *New Generation Comput.*, vol. 28, no. 2, pp. 111–112, 2010.
[5] "Condor dagman," Website, http://www.cs.wisc.edu/condor/dagman/.
[6] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Sci. Program.*, vol. 13, pp. 219–237, July 2005.

[7] Y. Zhao, M. Hategan, B. Clifford, I. T. Foster, G. von Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, and M. Wilde, "Swift: Fast, reliable, loosely coupled parallel computation," in *IEEE SCW*. Salt Lake City, Utah, USA: IEEE Computer Society, July 2007, pp. 199–206.
[8] I. Altintas, B. Ludaescher, S. Klasky, and M. A. Vouk, "Introduction to scientific workflow management and the kepler system," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, ser. SC'06. New York, NY, USA: ACM, 2006.
[9] L. Wang, D. Chen, and F. Huang, "Virtual workflow system for distributed collaborative scientific applications on grids," *Computers & Electrical Engineering*, vol. 37, no. 3, pp. 300–310, 2011.
[10] L. Wang, M. Kunze, and J. Tao, "Performance evaluation of virtual machine-based grid workflow system," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 15, pp. 1759–1771, 2008.
[11] L. Wang, G. von Laszewski, J. Tao, and M. Kunze, "Virtual data system on distributed virtual machines in computational grids," *IJAHUC*, vol. 6, no. 4, pp. 194–204, 2010.
[12] P. Missier, S. Soiland-Reyes, S. Owen, W. Tan, A. Nenadic, I. Dunlop, A. Williams, T. Oinn, and C. Goble, "Taverna, reloaded," in *Proceedings of the 22nd international conference on Scientific and statistical database management*, ser. SSDBM'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 471–481.
[13] L. Wang, G. von Laszewski, A. J. Younge, X. He, M. Kunze, J. Tao, and C. Fu, "Cloud computing: a perspective study," *New Generation Comput.*, vol. 28, no. 2, pp. 137–146, 2010.
[14] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107–113, January 2008.
[15] "Apache hadoop project," Web Page, http://hadoop.apache.org/.
[16] O. Tatebe, K. Hiraga, and N. Soda, "Gfarm grid file system," *New Generation Comput.*, vol. 28, no. 3, pp. 257–275, 2010.
[17] "Torque resource manager," Website, http://www.clusterresources.com/products/torque-resource-manager.php.
[18] "Distributed Resource Management Application API (DRMAA)," Website, http://drmaa.org/.



Lizhe Wang received his bachelor and master degree from Tsinghua University and doctoral degree from University. He currently is a professor at Chinese Academy of Science and a Chair professor at China University of Geosciences.

Dr. Jie Tao received her doctoral degree from Technical University of Munich. She currently is researcher at Karlsruhe Institute of Technology.

Dr. Yan Ma is a assistant Professor at Chinese Academy of Science. She received her Doctoral Degree from Chinese Academy of Science 2013.

Dr. Samee Kahn is an Assistant Professor of Electrical and Computer Engineering and an Adjunct Professor of Computer Science and Operations Research at the North Dakota State University (NDSU). He received my PhD (Computer Science) from the University of Texas and BS (Computer Systems Engineering) from the GIK Institute (Pakistan) in 2007 and 1999, respectively.

Joanna Kolodziej is a researcher at Institute of Computer Science, Cracow University of Technology, Porland. She received her Master and Ph.D. from Jagiellonian University in Cracow, Poland.

Dan Chen received his Doctoral degree from Nanyang Technological University, Singapore. He is currently a professor at China University of Geosciences.