

MapReduce Across Distributed Clusters for Data-intensive Applications

Lizhe Wang^{1,5}, Jie Tao², Holger Marten², Achim Streit², Samee U. Khan³, Joanna Kołodziej⁴ and Dan Chen⁵

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 Department of Mathematics and Computer Science, University of Bielsko-Biała, Poland

⁵ School of Computer, China University of Geosciences, P. R. China

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. Experiments of the G-Hadoop framework on distributed clusters show encouraging results.

Keywords: MapReduce, Hadoop, Data Intensive Computing

I. 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]–[3]. 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 petabytes of data in 2010 [4], [5]. 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.

Currently data-intensive workflow systems, such as Pegasus [6], Swift [7], 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 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. 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 II discusses background and related work of our research; Section III and Section IV presents the design and the performance evaluation of G-Hadoop. Finally, Section V concludes the paper and points out the future work.

II. BACKGROUND AND RELATED WORK

A. Cloud Computing

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 [8]. 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 paradigm and its open-sourced implementation – Hadoop have recognized as representative enabling technique for Cloud computing.

B. Distributed data intensive computing

To store, manage, access, and process vast amount of data represents a fundamental requirement and an immense challenge in order to satisfy needs to search, analyze, mine, and visualize the data and information. Data intensive computing is intended to address this need. In the research of data intensive computing, the study on massive data processing paradigm is of high interests for the research community [9], [10]. We focus in this paper an implementation of a data processing paradigm across multiple distributed clusters.

C. MapReduce paradigm

1) *MapReduce paradigm and Apache Hadoop project:* The MapReduce [12] 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 [13], 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 Apache Hadoop on Demand (HOD) [14] provides virtual Hadoop clusters over a large physical cluster. It uses the Torque resource manager to do node allocation. myHadoop

[15] is a system for provisioning on-demand Hadoop instances via traditional schedulers on HPC resources.

To the best of our knowledge, there is no existing implementation of the MapReduce paradigm across distributed multiple clusters.

D. Gfarm file system

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. Figure 1 shows the high-level architecture of the Gfarm file system.

In order to foster performance, the Gfarm file system separates metadata management from storage by leveraging a master/slave communication model. The single master node called Metadata Server (MDS) is responsible for managing the file system’s metadata such as file names, file locations and file’s access rights. The metadata server is also responsible for coordinating access to the files stored on the cluster.

The multiple slave nodes, referred to as Data Nodes (DN), on the other hand, are responsible for storing the raw file data on local hard disks using the provided local file system by the operating system of the slave node. A data node runs a daemon that coordinates the access to the files on the local file system. Direct access to the files is possible, but not encouraged due to risk of corrupting the file system.

In our work, we use Gfarm file system as a global distributed file system that supports MapReduce framework.

Gfarm file system does not use block based storage. Splitting files into blocks significantly increases the amount of metadata and hence, inherently impacts sacred latency and bandwidth in wide-area environments. To improve overall performance, the Gfarm file system uses a file based storage semantics. These performance improvements come at its costs: the maximal file size that can be managed by the Gfarm file system is limited by the capacity of disks used in Data Nodes of the distributed cluster.

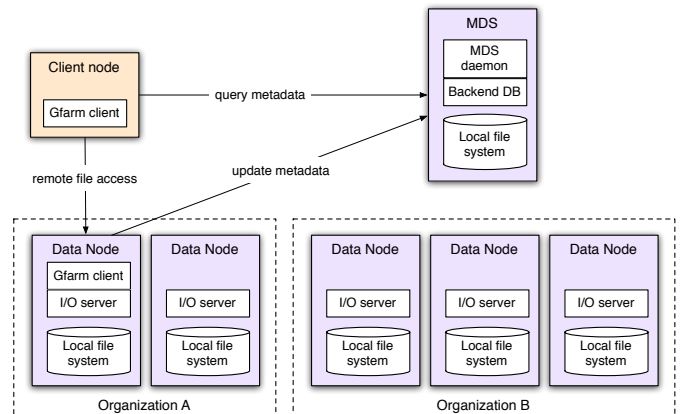


Fig. 1. The Gfarm file system architecture

E. Resource management for clusters

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.

III. G-HADOOP SYSTEM DESIGN

A. Target environment and development goals

Our target environment for G-Hadoop is a system of 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.

B. Architecture overview

The proposed architecture of G-Hadoop represents a master/slave communication model. Figure 2 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 we 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.

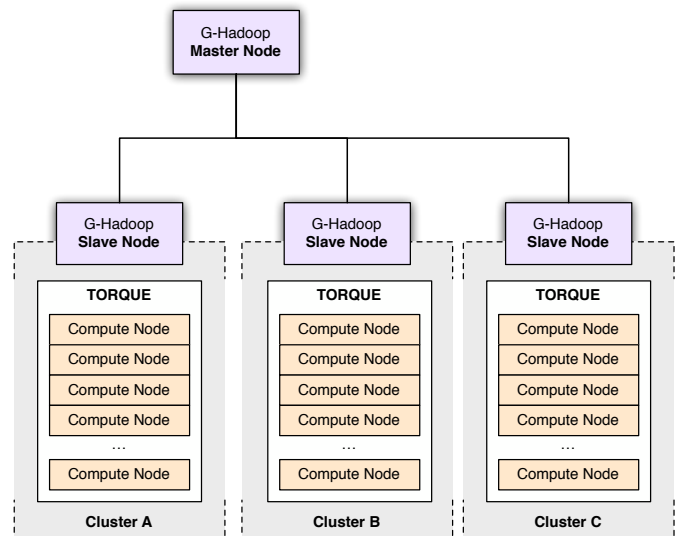


Fig. 2. Architecture Overview of G-Hadoop

C. 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.

D. 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 3 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 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.

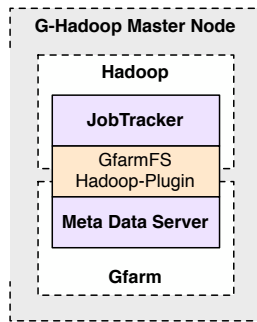


Fig. 3. Software components of the G-Hadoop master node

E. 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 4) consists of the following software components:

- *TaskTracker*: This server is a modification 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 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.

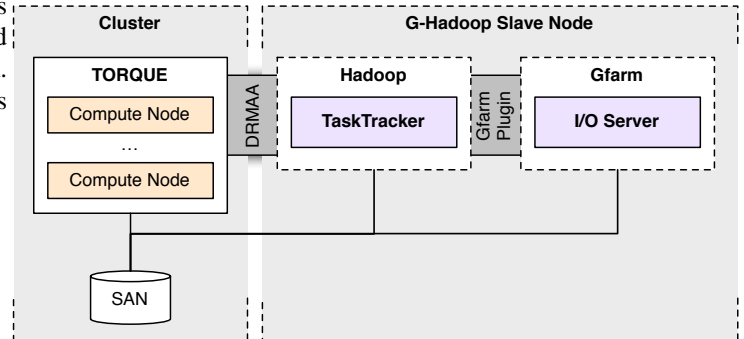


Fig. 4. Software components of G-Hadoops slave node

F. Job execution flow

Submitting a job to G-Hadoop is not different from submitting jobs to a traditional Hadoop cluster. Users write the MapReduce application for the desired job, the application is compiled and then run on the client node. When the program starts its execution the job is submitted to the *JobTracker* located on the G-Hadoop master node. The next stages of the job in the control flow managed by the *JobTracker* are described below. Figure 5 illustrates these stages:

- 1) *Job submission*: When the user starts a MapReduce application on a client node, the method *runJob()* is called (a). This method instantiates a *JobClient* (part of Hadoop's MapReduce stack) which is responsible for submitting the jobs to the system. The *JobClient* first contacts the *JobTracker* located on the G-Hadoop Master node and requests a unique ID for the new job (b). Upon a successful request the job client copies the MapReduce executable (JAR), its configuration including parameters and input files and additional resources to a designated working directory on the Gfarm file system (c). The *JobClient* then submits the job to the *TaskTracker* for execution (d).
- 2) *Job initialization*: On the G-Hadoop Master node the *JobTracker* initializes the job (a). The *JobTracker* then splits the job into smaller tasks by invoking the *generateInputSplit()* method of the job. This method can be implemented by the user. The default implementation contacts the Gfarm metadata server and requests all locations (including replicas) of the jobs input files (b). Since the Gfarm file system uses a file based approach blocks of different sizes representing the files along with the information on which cluster the files are located are returned. The number of map tasks is set to the

number of input files configured by the user. Each task is configured to use one of the file as its input data.

- 3) Task assignment: The *TaskTrackers* of the G-Hadoop slaves located on the participating clusters periodically ask the *JobTracker* for new tasks using the heartbeat message protocol. Based on the location information of the input files, tasks are assigned preferably to those clusters where the required input data is present. The *JobTracker* is able to answer the request with multiple (hundreds) new tasks using a single heartbeat message.
- 4) Task localization: When the *TaskTracker* receives a new task, it localizes the tasks executable and resources by copying the working directory from the Gfarm file system (a) to a network share on the cluster (b).
- 5) Task submission: After the task is localized on the cluster by the *TaskTracker*, the executable along with its working directory is submitted to the cluster scheduler using the DRMAA interface.
- 6) Task execution: At some point the cluster scheduler selects an idle compute node for the execution of the task (a). The compute node gets the jobs executable from the shared location in the jobs working directory (b). Required libraries (such as Hadoop and Gfarm) must be accessible on a dedicated shared location on the cluster. The job is executed by spawning a new JVM on the compute node and running the corresponding task with the configured parameters:
 - Map Task: If the task is a map task, it starts reading and processing the input files associated with the task. The output is written to a shared directory on the cluster and sorted afterwards (c).
 - Reduce Tasks: If the Task is a reduce task it starts fetching the outputs of the previously finished map tasks by contacting the *TaskTracker* which was responsible for the execution of the corresponding map task. If the *TaskTracker* is located at the same cluster as the reduce task, the files are read from the common shared location on the storage cluster. Otherwise the reduce task fetches the required map output using a HTTP request at the corresponding *TaskTracker* on the other cluster. The results of a reduce task are usually written to the Gfarm file system.

During the execution the tasks periodically report their health status to the *TaskTracker*. After the task finishes its execution, it reports its done status to the *TaskTracker* and exits.

- 7) Freeing the slot: The *TaskTracker* waits until the cluster scheduler executes the task, then frees the slot and is ready to progress with the next task.

IV. TESTS AND PERFORMANCE EVALUATION

This section discusses tests and performance evaluation for the G-Hadoop implementation.

A. Test 1: Performance comparison of Hadoop and G-Hadoop

1) *Test setup*: We have configured two clusters managed by Torque. Each node is equipped with 2 Dual Core AMD

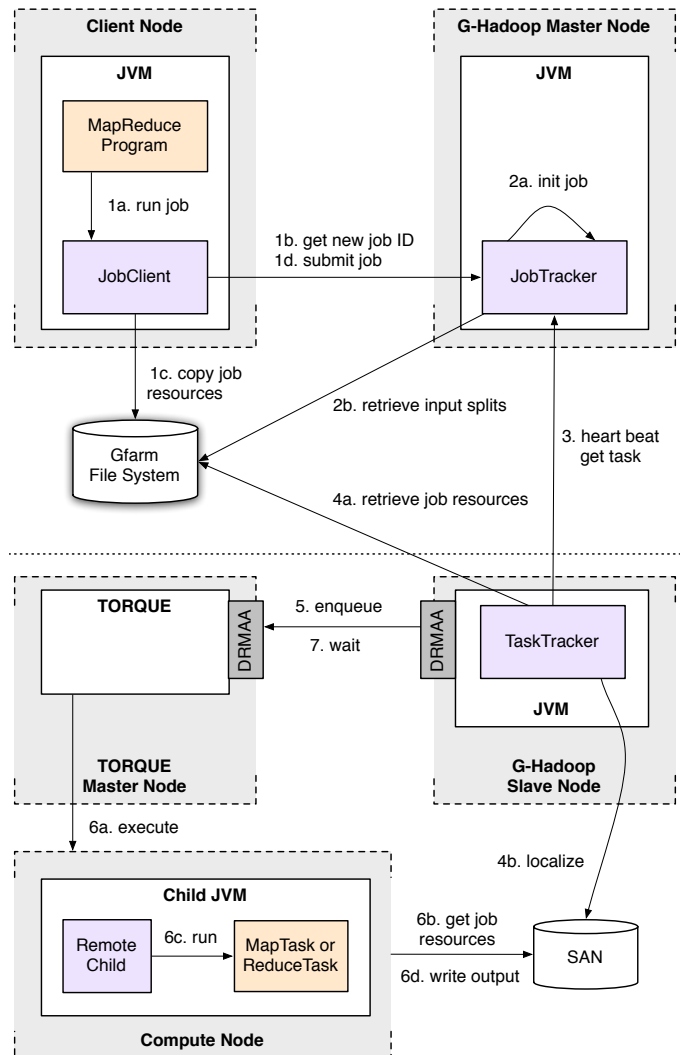


Fig. 5. Execution flow of a MapReduce job in G-Hadoop

Opteron™ Processor 270, 4 GB system memory, 1 Gigabit ethernet and 160 GB IDE Ultra ATA133. The operating system is CentOS 5.5 64 bit (kernel 2.6.18-194.26.1.el5).

For comparison purposes, we have performed the same experiments on the following deployments of Hadoop (Scenario A) and G-Hadoop (Scenario B):

- Scenario A is a deployment of an unmodified release of Hadoop version 0.21.0, basically in its default configuration:
 - 1 master node with a *JobTracker* and a *NameNode* (A.1),
 - 8 to 64 slave nodes with a *DataNode* and *TaskTracker* per slave node,
 - 2 slots per *TaskTracker*, (mapreduce.tasktracker.map.tasks.maximum = 2)
 - Hadoop installed and executed from a NFS share,
 - No additional optimization are applied.
- Scenario B is deployed using our prototype implementation of G-Hadoop with the following configuration:
 - 1 G-Hadoop master node (B.1) with 1 *JobTracker* and 1 Gfarm metadata server,

- 1 G-Hadoop slave node (B.2) with
 - 1 TaskTracker configured to run 16 to 128 tasks simultaneously,
 - 1 Gfarm *DataNode*,
 - 1 Torque master (*pbs_server*),
 - 1 Torque scheduler (default: *pbs_sched*),
- 8 to 64 Torque compute nodes each with 1 Torque slave (*pbs_mom*),
- Hadoop installed and executed from a NFS share.

We execute the MapReduce implementation of the Bailey-Borwein-Plouffe (BBP) [19] algorithm, which is part of the example benchmarks distributed within Apache Hadoop software. Before the job starts executing, the input splits for the configured number of tasks are calculated in order to reach a fair distribution of the workload.

We executed this benchmark on scenario (A) and (B) with cluster sizes from 16 to 64 compute nodes. The number of map tasks is set depending on the cluster size to the number of cores/slots (twice the number of nodes) and eightfold the number of cores/slots on the deployed system.

2) *Test results and performance evaluation:* Figure 6 shows our results for the “Sort” benchmark on a typical Hadoop environment with one TaskTracker on each compute node (Scenario A) and a G-Hadoop installation (Scenario B) with a single *TaskTracker* on the G-Hadoop slave node backed by multiple compute nodes managed by the cluster scheduler. Figure 6 shows that G-Hadoop can keep up with the performance of the default Hadoop installation by a sheer margin of maximum of 6%. Figure 6 also shows that increasing the number of compute nodes in the cluster results in a slight decrease of performance compared to the default Hadoop installation deployed with the same number of compute nodes. However, we believe that these penalties correlate with another observation we made during our tests: Tasks are submitted to the cluster scheduler using a sequential approach on the TaskTracker. We observed that submitting hundreds of tasks at once requires up to one minute until the cluster scheduler accepts the last task into its queue.

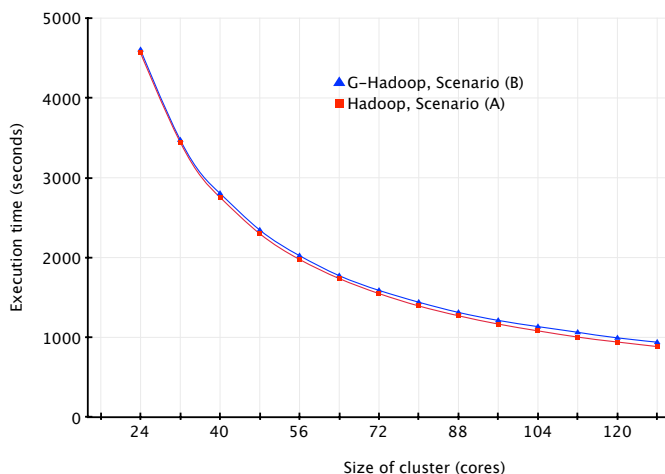


Fig. 6. Execution times of BBP using 1 map task per core

Figure 7 shows the results of another experiment similar

to the previous, but this time configured with eight times as many map tasks as cores available in the cluster. In this experiment the single TaskTracker on the G-Hadoop slave node (B2) must periodically make sure to keep the queue of the cluster scheduler busy. Figure 7 shows a notable performance decrease to up to 13% compared to the equally sized Hadoop.

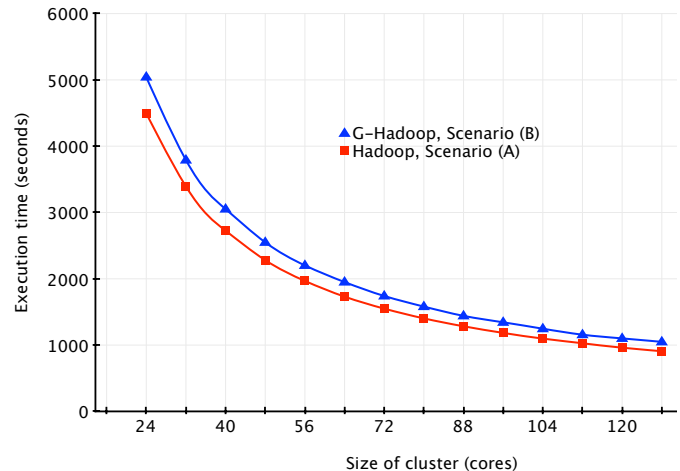


Fig. 7. Execution times of BBP using 8 map tasks per core

Figure 8 and Figure 9 show the bandwidth measured on the Hadoop master node (A.1) and on the G-Hadoop master node (B.1) during the execution of the job. Periodical peaks can be observed in both experiments. Noteworthy is that the peaks of incoming data are about half the size the peaks reached in the default Hadoop deployment. Furthermore, the bandwidth used on the Hadoop master (A.1) shows a lot more traffic between 50 and 100 KB/S. These observations can be traced back to the fact that every TaskTracker on the Hadoop cluster (A) sends a heartbeat message to the *JobTracker*. In comparison, the single TaskTracker on the G-Hadoop cluster (B) sends accumulated information about all 128 concurrently running tasks instead of only 2.

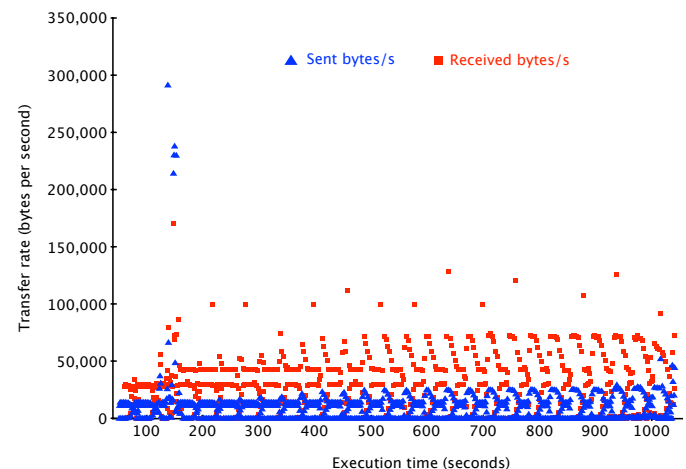


Fig. 8. Bandwidth usage on Hadoop master node (A.1)

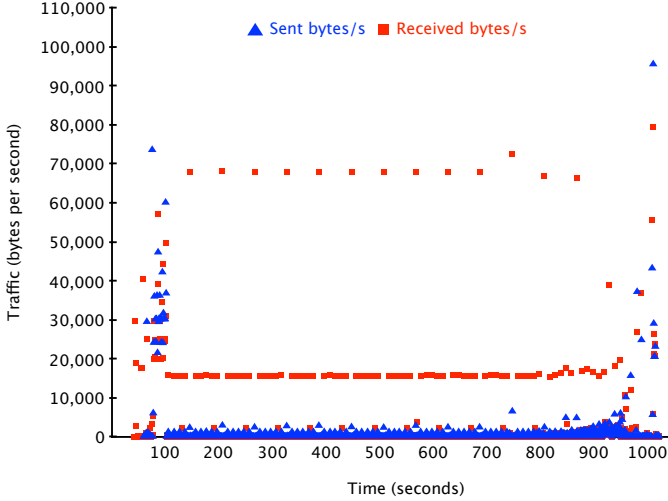


Fig. 9. Bandwidth usage on G-Hadoop master node (B.1)

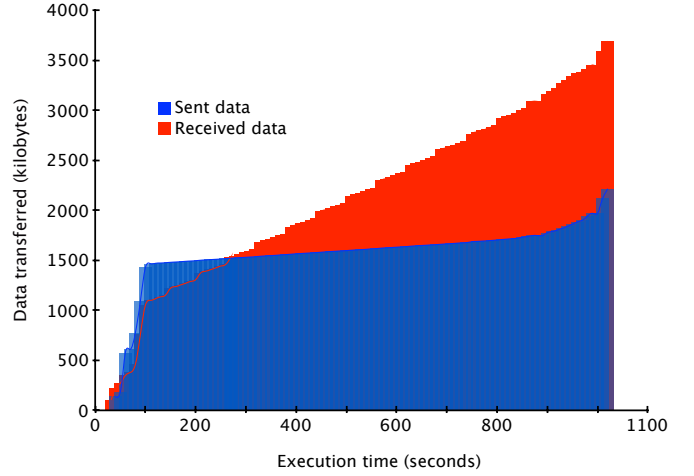


Fig. 11. Accumulated traffic on Hadoop master node (B.1)

Figure 11 shows the accumulated traffic measured on the master node of G-Hadoop (B.1). The combined amount of data required for the execution of the BBP job measures about 5.5 megabytes. Executing the same job on a Hadoop cluster with the same size, in contrast, requires about 32 MB and therefore about six times as much data to be transferred over the wide area network. Figure 10 depicts these measurements.

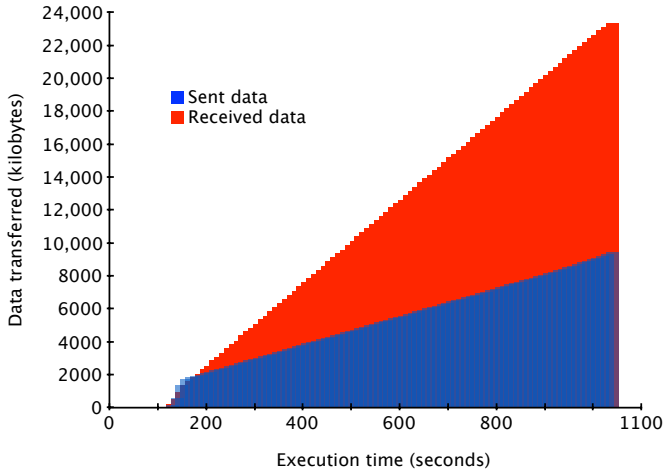


Fig. 10. Accumulated traffic on Hadoop master node (A.1)

B. Test 2: Performance evaluation of G-Hadoop

1) *Test setup:* We setup two identical clusters with the Amazon Elastic Compute Cloud. Table I shows the node setup of the clusters.

We deploy the test on each cluster with configuration described in Scenario B (see Section IV-A). Each test cluster is setup with 4, 8, 12, 16 nodes and every cluster node is deployed with 8 mappers.

We used the Hadoop Sort benchmark [20] to evaluate our implementation. In Hadoop Sort benchmark, the entire test dataset goes through the shuffle stage, which requires

TABLE I
NODE CONFIGURATION FOR G-HADOOP PERFORMANCE EVALUATION

Item	Configuration
OS	Cent OS 5.5
AMI	centos5.5-minimum-i386-ebs (ami-1819eb71)
Processor	Intel(R) Xeon(R) CPU E5507 @ 2.27GHz
Memory	1.7 GB
Harddisk	20 GB

the network communication between slaves. The Sort results generate the same amount of data and save back to the HDFS in the reduce step. Therefore the Hadoop Sort benchmark is a suitable benchmark to evaluate the Hadoop deployment, network and HDFS performance.

Figure 12 shows the number of active mapper tasks within one cluster (node no. = 4) when executing the Sort benchmark (2048 M). As every node is deployed with 8 mappers, the maximum number of active mapper tasks inside one cluster is 32. In Figure 12, the running mapper task number increases quickly to reach its maximum, then stays approximately constant during the map stage. After entering the reduce stage, the running mapper task numbers decrease to almost 0.

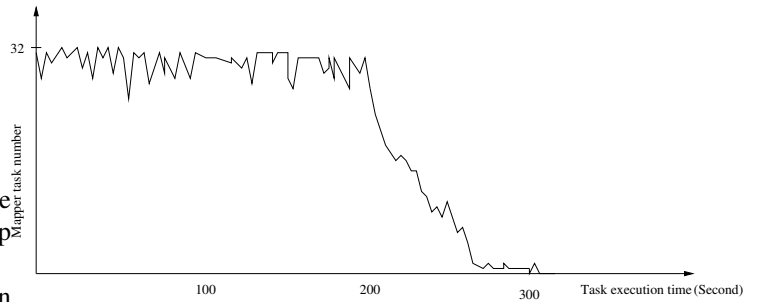


Fig. 12. Mapper task number during Sort benchmark execution with G-Hadoop

Figure 13 shows Sort benchmark task execution time with G-Hadoop on the test bed with various sort data set. We can see that G-Hadoop scale well as the input workload (data

size for sort) increases. Figure 14 shows Sort benchmark task execution time with G-Hadoop on the test bed with cluster node number. Figure 14 indicates that G-Hadoop has a similar behavior with Test 1 in Section IV-A when cluster size scales.

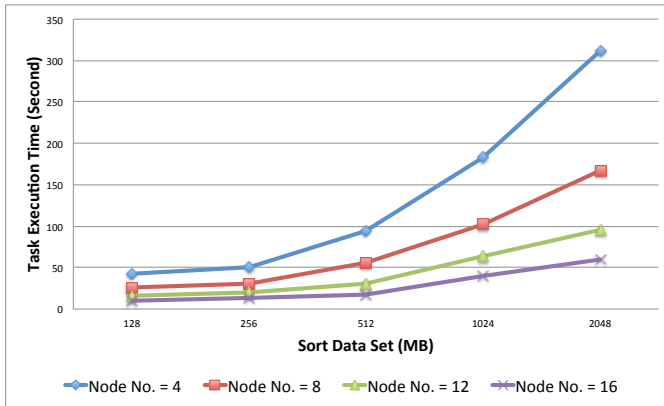


Fig. 13. Task execution time of Sort benchmark with various input data set

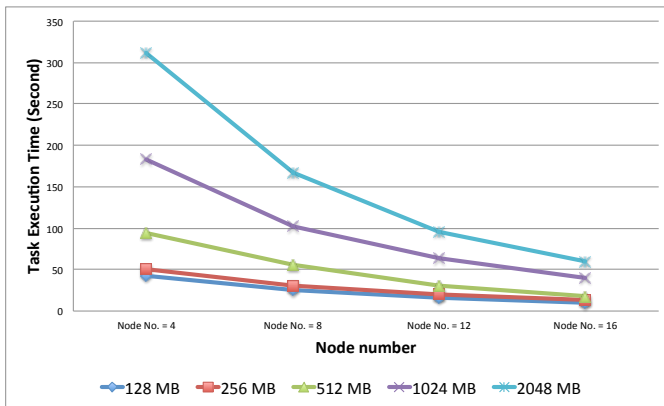


Fig. 14. Task execution time of Sort benchmark with G-Hadoop clusters

V. CONCLUSION AND FUTURE WORK

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 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. We have run several experiments on our prototype. The results show that the G-Hadoop has a comparable performance to the Apache Hadoop clusters and scales nearly linear with respect to the number of nodes in the cluster.

To make G-Hadoop fully functional, next step we plan to implement the coordinated hierarchical task scheduling algorithms and security services for the G-Hadoop framework.

ACKNOWLEDGEMENT

Dr. Lizhe Wang's work is funded by "One-Hundred Talents Program" of The Chinese Academy of Sciences.

REFERENCES

- [1] F. Berman, "Got data?: a guide to data preservation in the information age," *Commun. ACM*, vol. 51, pp. 50–56, December 2008. [Online]. Available: <http://doi.acm.org/10.1145/1409360.1409376>
- [2] "Data intensive computing." [Online]. Available: http://en.wikipedia.org/wiki/Data_Intensive_Computing
- [3] 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.
- [4] G. Brumfiel, "High-energy physics: Down the petabyte highway," *Nature*, no. 7330, pp. 282–283, January 2011.
- [5] L. Wang and C. Fu, "Research advances in modern cyberinfrastructure," *New Generation Comput.*, vol. 28, no. 2, pp. 111–112, 2010.
- [6] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laitly, 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. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1239649.1239653>
- [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] 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.
- [9] D. Chen, L. Wang, G. Ouyang, and X. Li, "Massively parallel neural signal processing on a many-core platform," *Computing in Science and Engineering*, 2011.
- [10] X. Yang, L. Wang, and G. von Laszewski, "Recent research advances in e-science," *Cluster Computing*, vol. 12, no. 4, pp. 353–356, 2009.
- [11] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107–113, January 2008. [Online]. Available: <http://doi.acm.org/10.1145/1327452.1327492>
- [12] "Apache hadoop project," Web Page, <http://hadoop.apache.org/>.
- [13] "Apache Hadoop on Demand (HOD)." [Online]. Available: http://hadoop.apache.org/common/docs/r0.21.0/hod_scheduler.html
- [14] S. Krishnan, M. Tatineni, and C. Baru, "myhadoop – hadoop-on-demand on traditional hpc resources," University of California, San Diego, Tech. Rep., 2011.
- [15] O. Tatebe, K. Hiraga, and N. Soda, "Gfarm grid file system," *New Generation Comput.*, vol. 28, no. 3, pp. 257–275, 2010.
- [16] "Torque resource manager," Website. [Online]. Available: <http://www.clusterresources.com/products/torque-resource-manager.php>
- [17] "Distributed Resource Management Application API (DRMAA)." [Online]. Available: <http://drmaa.org/>
- [18] D. H. Bailey, "The bbp algorithm for pi," Sep. 2006. [Online]. Available: <http://crd.lbl.gov/~dhbailey/dhbpapers/bbp-alg.pdf>
- [19] "Hadoop sort benchmark," Website. [Online]. Available: <http://wiki.apache.org/hadoop/Sort>