Future Generation Computer Systems 29 (2013) 739–750

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

G-Hadoop: MapReduce across distributed data centers for data-intensive computing

Lizhe Wang [a,b,*], Jie Tao [c], Rajiv Ranjan [d], Holger Marten [c], Achim Streit [c], Jingying Chen [e], Dan Chen [a,**]

[a] School of Computer, China University of Geosciences, PR China
[b] Center for Earth Observation and Digital Earth, Chinese Academy of Sciences, PR China
[c] Steinbuch Center for Computing, Karlsruhe Institute of Technology, Germany
[d] ICT Centre, CSIRO, Australia
[e] National Engineering Center for E-Learning, Central China Normal University, PR China

ARTICLE INFO

Article history:
Received 15 November 2011
Received in revised form
1 September 2012
Accepted 3 September 2012
Available online 3 October 2012

Keywords:
Cloud computing
Massive data processing
Data-intensive computing
Hadoop
MapReduce

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 is 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.

1. Introduction

The rapid growth of the 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 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 center 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 tackling 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 DAGMan [6], Pegasus [7], Swift [8], Kepler [9], Virtual Workflow [10,11], Virtual Data System [12] and Taverna [13], 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

* Corresponding author at: Center for Earth Observation and Digital Earth, Chinese Academy of Sciences, PR China.
** Corresponding author.
E-mail addresses: Lizhe.Wang@gmail.com (L. Wang), danjj43@gmail.com (D. Chen).

0167-739X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2012.09.001

require large data transfer between tasks, sometimes 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 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. The key differentiators between traditional Hadoop and the proposed G-Hadoop framework includes

- Unlike the Hadoop framework which can schedule data processing tasks on nodes that belong to a single cluster, G-Hadoop can schedule data processing tasks across nodes of multiple clusters. These clusters can be controlled by different organizations;
- By duplicating map and reduce tasks across multiple cluster nodes, G-Hadoop presents a more fault-tolerant data processing environment as it does not rely on nodes of a single cluster;
- G-Hadoop offers access to a larger pool of processing and storage nodes.

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 presents the G-Hadoop design. The G-Hadoop attributes and features are discussed in Section 4 and the G-Hadoop performance is tested and evaluated in Section 5. Finally Section 6 concludes the paper and points out the future work.

2. Background and related work

2.1. 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 [14,5,3,15,16]. Conceptually, users acquire computing platforms, or IT infrastructures, from computing Clouds and execute their applications inside them. 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 has been recognized as a representative enabling technique for Cloud computing.

2.2. Distributed data-intensive computing

To store, manage, access, and process vast amounts 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 these needs. In the research of data intensive computing, the study on massive data processing paradigm is of high interest for the research community [17–19]. Our research in this paper is on the implementation of a data processing paradigm across multiple distributed clusters.

There have been some successful paradigms and models for data intensive computing, for example, All-Pairs [20], Sector/Sphere [21], DryadLINQ [22], and Mortar [23]. Among aforementioned technologies, the MapReduce [24] is a widely adopted massive data processing paradigm, which is introduced in the next section. Our work implements the MapReduce paradigm across distributed clusters.

2.3. MapReduce and Hadoop

2.3.1. MapReduce paradigm

The MapReduce [24] programming model is inspired by two main functions commonly used 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 most popular implementation of the MapReduce model is the Hadoop framework [25], which allows applications to run on large clusters built from commodity hardware. The Hadoop framework transparently provides both reliability and data transfer. Other MapReduce implementations are available for various architectures, such as for CUDA [26], in a multicore architecture [27], in FPGA platforms [28], for a multiprocessor architecture [29], in a large-scale shared-memory system [30], in a large-scale cluster [31], in multiple virtual machines [32], in a.Net environment [33], in a streaming runtime environment [34], in a Grid environment [35], in an opportunistic environment [36], and in a mobile computing environment [37].

The Apache Hadoop on Demand (HOD) [38] provides virtual Hadoop clusters over a large physical cluster. It uses the Torque resource manager to do node allocation. myHadoop [39] 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.

2.3.2. Hadoop

The MapReduce programming model is designed to process large volumes of data in parallel by dividing the *Job* into a set of independent *Task*s. The *Job* referred to here as a full MapReduce program, which is the execution of a *Mapper* or *Reducer* across a set of data. A *Task* is an execution of a *Mapper* or *Reducer* on a slice of data. So the MapReduce *Job* usually splits the input data set into independent chunks, which are processed by the *map* tasks in a completely parallel manner.

The Hadoop MapReduce framework consists of a single Master node that runs a *Jobtracker* instance which accepts *Job* requests from a client node and *Slave node*s each running a *TaskTracker* instance. The *Jobtracker* assumes the responsibility of distributing the software configuration to the Slave nodes, scheduling the job's component tasks on the TaskTrackers, monitoring them and re-assigning tasks to the *TaskTracker*s when they failed. It is also responsible for providing the status and diagnostic information to the client. The *TaskTracker*s execute the tasks as directed by the *JobTracker*. The *TaskTracker* executes tasks in separate java processes so that several task instances can be performed in parallel at the same time. Fig. 1 depicts the different components of the MapReduce framework.

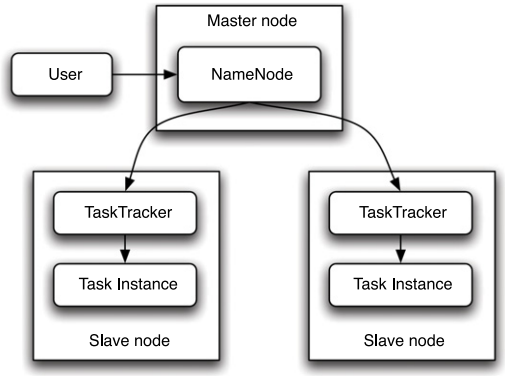


Fig. 1. Hadoop MapReduce.

Fig. 2 illustrates the high-level pipeline of the Hadoop MapReduce. The MapReduce input data typically come from the input files loaded into the HDFS. These files are evenly distributed across all the nodes in the cluster. In Hadoop, computer nodes and data nodes are all the same, meaning that the MapReduce and HDFS run on the same set of nodes. At the mapping phase, the input file is divided into independent InputSplits and each split of these Splits describes a unit of work that comprises a single map task in the MapReduce job. The map tasks are then assigned to the nodes in the system based on the physically residence of the input file splits. Several map tasks can be assigned to an individual node, which attempts to perform as many tasks in parallel as it can. When the mapping phase has completed, the intermediate outputs of the map tasks are exchanged between all nodes; and they are also the input of the reduction tasks. This process of exchanging the map intermediate outputs is known as the shuffling. The reduce tasks are spread across the same nodes in the cluster as the mappers. The output of the reduce tasks is stored locally on the slave node.

2.3.3. HDFS

The HDFS has some desired features for massive data parallel processing, such as: (1) work in commodity clusters with hardware failures, (2) access with streaming data, (3) deal with large data sets, (4) employ a simple coherency model, and (5) portable across heterogeneous hardware and software platforms.

The HDFS has a master/slave architecture (Fig. 3). A HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of *DataNode*s, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of *DataNode*s. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to *DataNode*s. The *DataNode*s are responsible for serving read and write requests from the file systems clients. The *DataNode*s also perform block creation, deletion, and replication upon instruction from the *NameNode*.

2.4. Gfarm file system

The Gfarm file system [40] 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

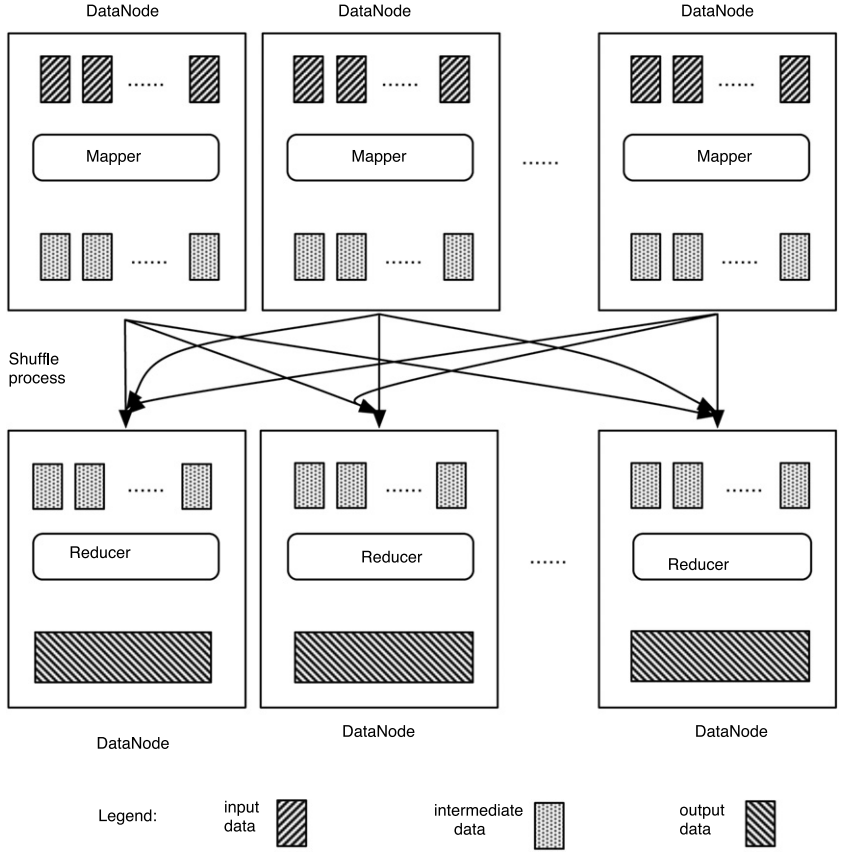


Fig. 2. Hadoop high level data flow.

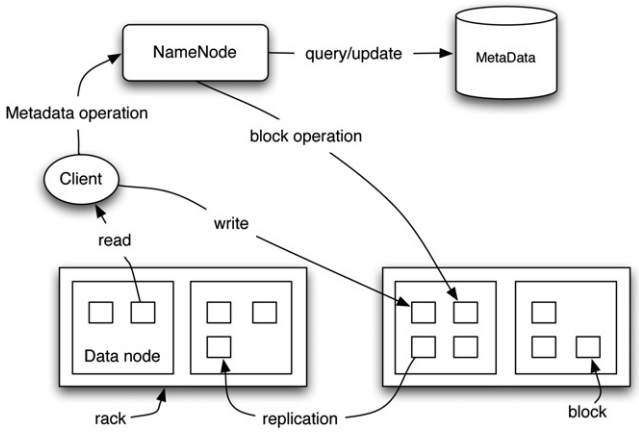


Fig. 3. HDFS architecture.

Gfarm file system. Fig. 4 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 a Metadata Server (MDS) is responsible for managing the file system's metadata such as file names, file locations and a 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 the risk of corrupting the file system.

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

The 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 a cost: 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.

2.5. 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 [41] as a cluster DRMS. Distributed Resource Management Application API (DRMAA) [42] 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 a 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. Fig. 5 shows an overview of 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

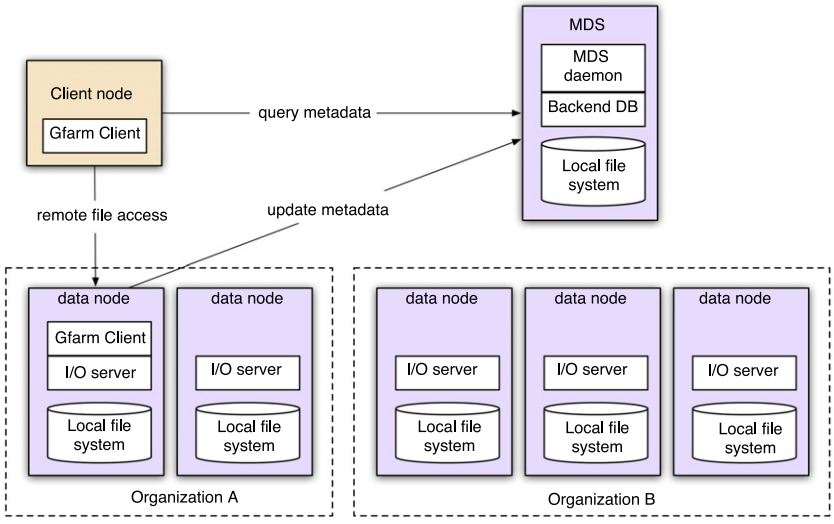


Fig. 4. The Gfarm file system architecture.

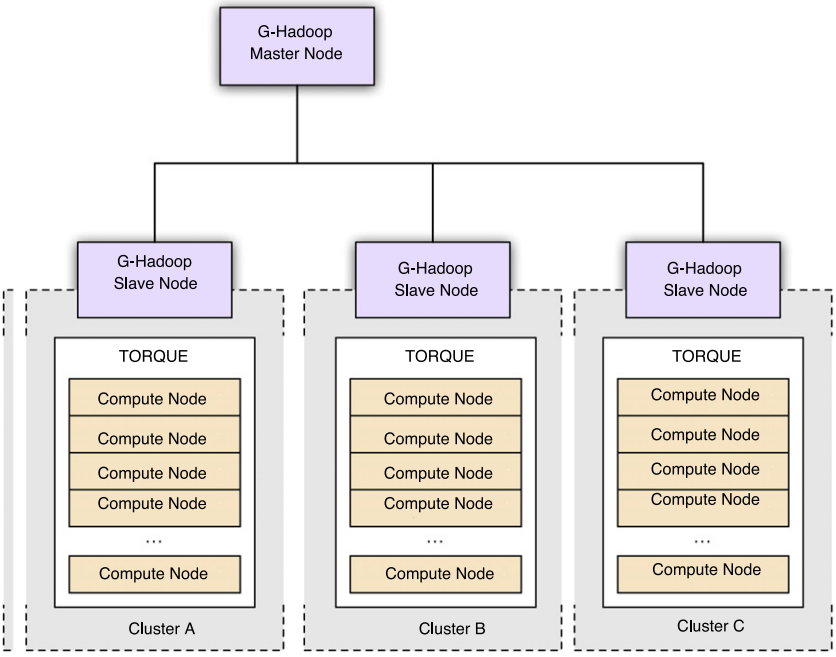


Fig. 5. Architecture overview of G-Hadoop.

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. Our G-Hadoop framework respects the autonomy of the local cluster scheduler, as G-Hadoop does not require any changes to the existing cluster system architecture. Software components enabling G-Hadoop Slave node functionalities only need to be installed on the cluster scheduler, hence requiring no changes to the existing cluster configuration.

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 Fig. 6 is composed of the following software components:

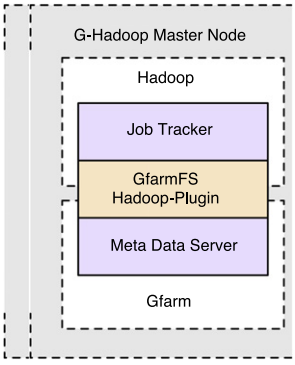


Fig. 6. Software components of the G-Hadoop master node.

- 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 keeping 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.

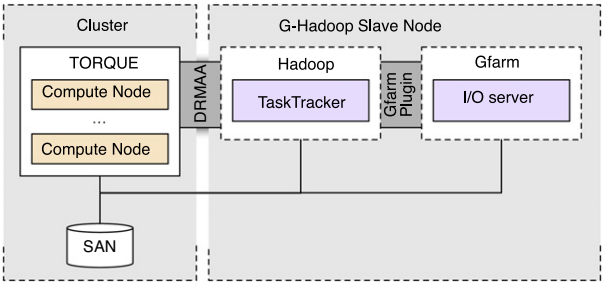


Fig. 7. Software components of G-Hadoop's slave node. The slave node acts as a bridge between the G-Hadoop master node and compute nodes of the clusters. A node in the cluster is chosen for installing the G-Hadoop slave components including TaskTracker and the Gfarm I/O server. The slave node does not perform any computation tasks, it is only responsible for submitting task to the queue of the cluster scheduler via the DRMAA interface. To improve resilience to failures, multiple slave nodes per cluster can be created.

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 Fig. 7) consists of the following software components:

- *TaskTracker*. This server is an adapted version of the 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 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 job's 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.

3.6. 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. Fig. 8 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 job's 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 files as its input data.
3. Task assignment: The *TaskTracker*s 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.

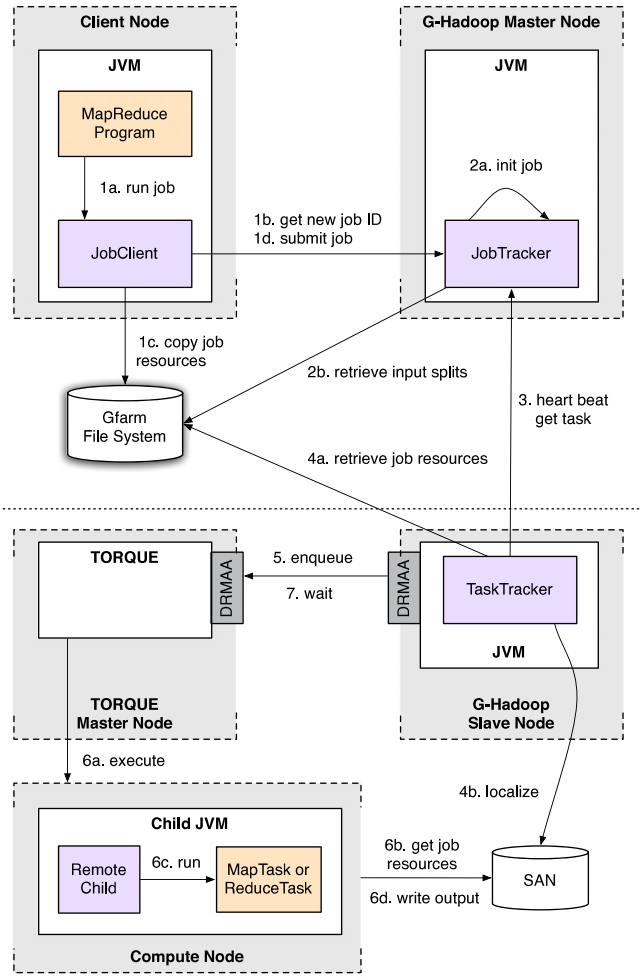


Fig. 8. Execution flow of a MapReduce job in G-Hadoop.

4. Task localization. When the *TaskTracker* receives a new task, it localizes the task's 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 job's executable from the shared location in the job's 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.

4. Discussion

4.1. Scalability

The MapReduce framework is known for its ability to scale linearly when it comes to processing of large data sets. Due to the fact that jobs can be split into a large amount of small and independent tasks, building Hadoop clusters with about 3800 nodes (30,400 cores) is possible nowadays. G-Hadoop is designed to build upon this approach on a global scale. The scalability of the G-Hadoop is mostly affected by the communication between the *JobTracker* and the *TaskTracker*s and the performance of the Gfarm Grid file system backed by the parallel file system on the cluster.

4.1.1. Communication between JobTracker and TaskTracker

In order to enable scalability in wide-area environments, G-Hadoop takes a different approach from the Hadoop implementation. In G-Hadoop each *TaskTracker* is responsible for a cluster with hundreds or thousands of compute nodes instead of a single node. The number of *TaskTracker*s therefore only grows with the number of the affiliated clusters. The number of compute nodes, however, grows with each additional cluster by the number of compute nodes at the cluster. The hierarchical scheduling approach taken by G-Hadoop enables a compound heartbeat for hundreds or thousands of slots and encourage better scalability characteristics in wide-area environments.

4.1.2. Performance of the distributed file system

The Gfarm file system is capable of scaling to thousands of data nodes and thousands of clients. Although a minimal G-Hadoop deployment only employs a single I/O server per cluster, the number of installed I/O servers is not limited by G-Hadoop's architecture and may be increased as desired in order to improve performance of the distributed file system.

In contrast to a traditional Hadoop cluster where data located on the local file system is preferably used for data intensive tasks, G-Hadoop's compute nodes do not rely on local storage when reading input data. Instead, the compute nodes retrieve their input data over the high performance network through the Gfarm I/O server and induce a non-neglectable performance hit. To tackle this problem we can leverage the fact that the Gfarm I/O servers federate the underlying file system of the node the I/O server is running on. Configuring the Gfarm I/O server to use a directory on the parallel cluster file system and retrieving the local data directly over the network (i.e., bypassing the I/O server) in read only mode may significantly improve the performance for data intensive scenarios.

4.2. Fault tolerance

The G-Hadoop is built on Hadoop for executing MapReduce jobs and the Gfarm file system as the underlying distributed file system. Both components employ a master/slave communication model where both masters induce a single point of failure. This section discusses possible failures of individual components in G-Hadoop and their impact on the G-Hadoop's availability.

4.2.1. Task

In G-Hadoop hundreds of tasks are executed in child JVMs running remotely on different compute nodes managed by a single *TaskTracker*. A running task periodically informs the *TaskTracker* about its current status. When the *TaskTracker* detects a failed task it reports the failure to the *JobTracker* during the next heartbeat. The default behavior of the *JobTracker* in this case is to re-schedule the task on another *TaskTracker* rather than the one where the task failed. If the task fails multiple times (on different *TaskTracker*s) the whole job is assumed to be error-prone and typically fails.

This behavior is reasonable in traditional instances of Hadoop with thousands of *TaskTracker*s and only a small number of parallel tasks per *TaskTracker*. However, re-scheduling the failed task on a different *TaskTracker* in G-Hadoop would imply the execution of the task on another cluster most likely with off-site input data. G-Hadoop can tackle this problem by allowing re-scheduling of tasks to the same *TaskTracker* preferable. A more sophisticated solution would be to allow the *TaskTracker* to reschedule failed tasks transparently instead of reporting them to the *JobTracker*.

4.2.2. Metadata server

The metadata server manages file names, directories and other essential metadata of all files in the system. If this server goes down the whole file system cannot be accessed. Fortunately the probability of the failure of a single machine is low and can be reduced to a minimum by using more reliable hardware architectures. In addition, the metadata server manages a backup of its database on persistent storage. In case of a blackout for example, this backup can be used to restore the metadata that is usually held completely in memory for reasons of performance.

4.2.3. JobTracker

The *JobTracker* also represents a single point of failure in the G-Hadoop architecture. Since the *JobTracker* runs on a single node the chance of a failure is low and can be reduced using the same strategies as for the metadata server. Unfortunately Hadoop's *JobTracker* does not provide any fail-over mechanisms yet. If the *JobTracker* fails, all running jobs will fail and must be restarted by the user again.

4.2.4. TaskTracker

The availability of the *TaskTracker* can be improved by increasing the number of *TaskTracker*s used per cluster. Since *TaskTracker*s periodically report their health status, the *JobTracker* is able to detect a failed *TaskTracker* after the absence of the heartbeat for a configurable amount of time. The *TaskTracker* is then marked as failed by the *JobTracker* and the currently running tasks are re-scheduled on another *TaskTracker*.

4.2.5. Gfarm I/O server

The impact of the failure of the Gfarm I/O server and its hosting node depends on the replication factor that was configured by the administration. Typically the number of replicas is set to at least a number of two, so in case of a failure of a single I/O server, the affected files can be restored from another location. We encourage running multiple I/O servers on multiple nodes per cluster in order to foster availability of the data and performance.

4.2.6. Cluster scheduler

The availability of a cluster scheduler usually depends on the configuration made by the organization and is independent of G-Hadoop. If the cluster scheduler fails, the individual tasks submitted by the *TaskTracker* to the cluster scheduler fail. In this case, the same mechanism as for the failing *TaskTracker* is triggered: The *JobTracker* marks the *TaskTracker* as failed and re-schedules all the tasks to other *TaskTracker*s.

4.2.7. Security model

Though G-Hadoop is a extension of the Hadoop MapReduce Framework, it cannot simply reuse the user authentication and task submission mechanism of Hadoop. This is mainly due to fact that the security framework in Hadoop is designed for a single cluster and hence does not cater well to the new challenges raised by the distributed Grid (multi-cluster) environment. These new challenges include heterogeneous resource control policies, distributed organization, and multiple types of cluster resources. In G-Hadoop, for example, an individual SSH connection has to be built between the user and each cluster to which his task is submitted. In addition, with the Hadoop security approach a user must log on to each cluster in order to be authenticated before being capable of deploying his tasks. Hence, the number of SSH connections per user will grow as $O(N)$ with the traditional Hadoop security framework, where N is the number of clusters in the system. Clearly, this is a tedious task for the users.

To overcome the aforementioned challenges posed by the Grid environment, in our recent work [43] we designed and implemented a new security approach for the G-Hadoop framework. Here we will give a brief overview of the security model, however interested readers may refer to the following paper [43] for more details. The security model has the following features:

- A single sign-on process with a user name and password. A user logs on to G-Hadoop with his user name and password. Following that, all available cluster resources are made accessible to the user via the same authentication and authorization module. The procedure of being authenticated and authorized across multiple clusters is performed automatically by the security framework behind the scene.
- Privacy of user's credentials. The user's credentials, such as authentication (user-name, password, etc.) information, is invisible to the slave nodes. Slave nodes accept tasks, which are assigned to them by the master node, without being aware of a user's credentials.
- Access control. The security framework protects nodes of distributed cluster resources from abuse by users. Only users who possess appropriate credentials have the right to access the cluster resources via an SSH connection.
- Scalability. A cluster can be easily integrated or removed from the execution environment without any change of the code on the slave nodes or any modification of the security framework itself.
- Immutability. The security framework does not change the existing security mechanism of the clusters. Users of a cluster can still rely on their own authentication profiles for getting access to the clusters.
- Protection against attacks. The new security framework protects the system from different common attacks and guarantees the privacy and security by exchanging sensitive information such as encrypted user-name and password. It is also capable of detecting the fraudulent user who may try to fake credentials.

5. Tests and performance evaluation

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

5.1. Test 1: performance comparison of Hadoop and G-Hadoop

5.1.1. Test setup

We have configured a cluster of test bed managed by Torque. Each node is equipped with 2 Dual Core AMD Opteron^{tm} Processor 270, 4 GB system memory, 1 GB 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),
  - 8–64 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 is 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 16–128 tasks simultaneously,
    - 1 Gfarm *DataNode*,
    - 1 Torque master (*pbs_server*),
    - 1 Torque scheduler (default: *pbs_sched*),
  - 8–64 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) [44] 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 scenarios (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 eight times the number of cores/slots on the deployed system.

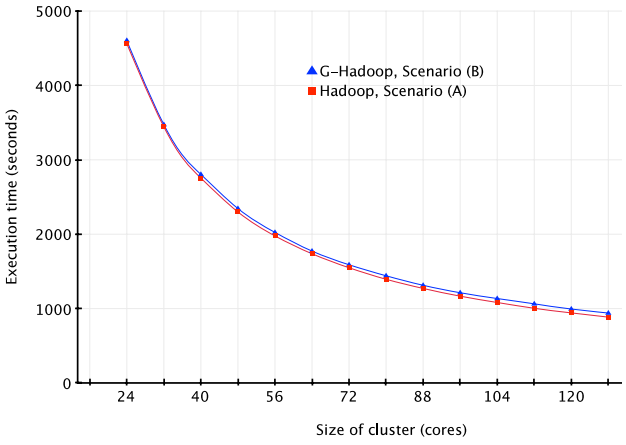


Fig. 9. Execution times of BBP using 1 map task per core.

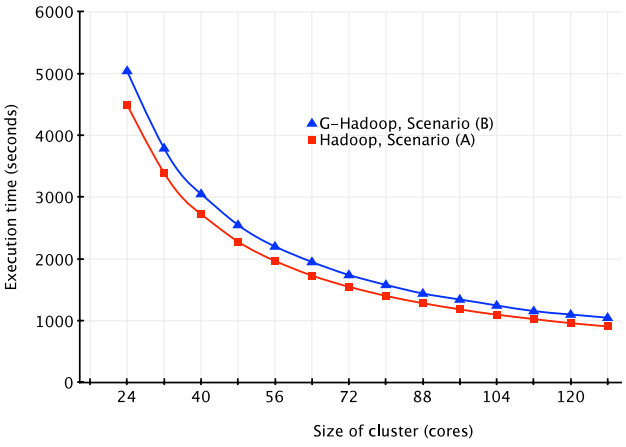


Fig. 10. Execution times of BBP using 8 map tasks per core.

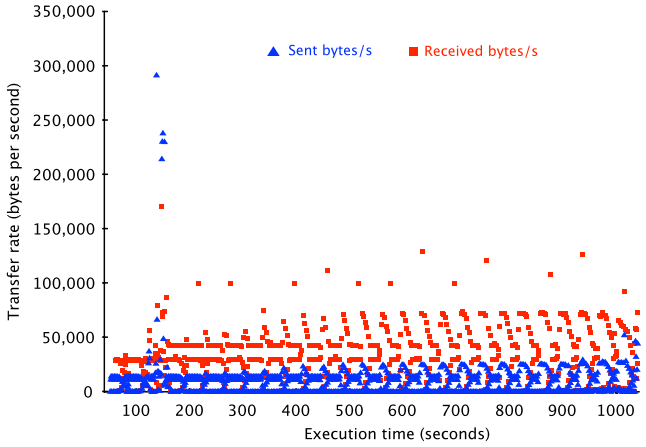


Fig. 11. Bandwidth usage on Hadoop master node (A.1).

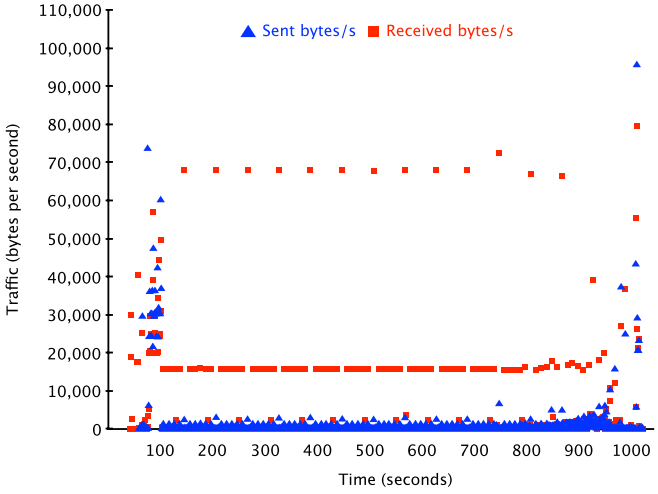


Fig. 12. Bandwidth usage on G-Hadoop master node (B.1).

5.1.2. Test results and performance evaluation

Fig. 9 shows our measurements for running the "sort" benchmark tasks 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. Fig. 9 shows that G-Hadoop can keep up with the performance of the default Hadoop installation by a sheer margin of max. 6%. Fig. 9 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. 10 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. Fig. 10 shows a notable performance decrease to up to 13% compared to the equally sized Hadoop.

Figs. 11 and 12 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. 14 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 MB. 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. Fig. 13 depicts these measurements.

5.2. Test 2: performance evaluation of G-Hadoop

5.2.1. Test setup

We setup the same test bed described in Section 5.1.1 with the Amazon Elastic Compute Cloud.

We deploy the test on the test bed with the configuration described in Scenario B (see Section 5.1). Each test cluster is setup with 4, 8, 12, or 16 nodes and every cluster node is deployed with 8 mappers.

We used the Hadoop Sort benchmark [45] to evaluate our implementation. In the Hadoop Sort benchmark, the entire test

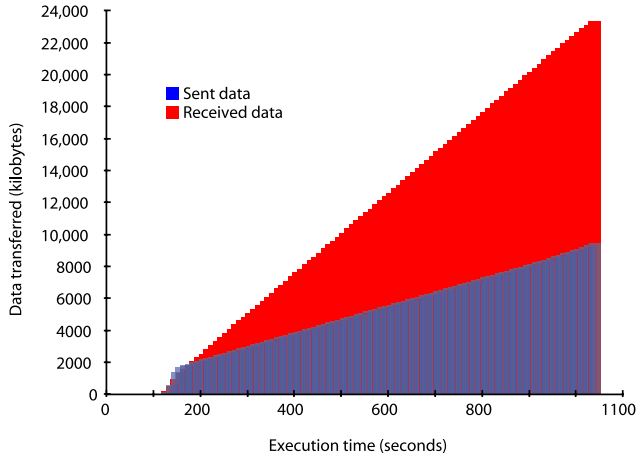


Fig. 13. Accumulated traffic on Hadoop master node (A.1).

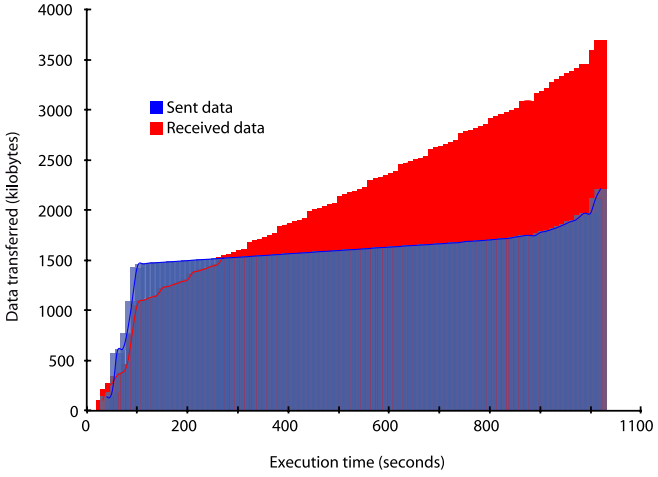


Fig. 14. Accumulated traffic on Hadoop master node (B.1).

Fig. 15. Mapper task number during Sort benchmark execution with G-Hadoop.

dataset goes through the shuffle stage, which requires 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.

Fig. 15 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 Fig. 15, 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. 16 shows the Sort benchmark task execution time with G-Hadoop on the test bed with various sort data sets. We can see that G-Hadoop scales well as the input workload (data size for

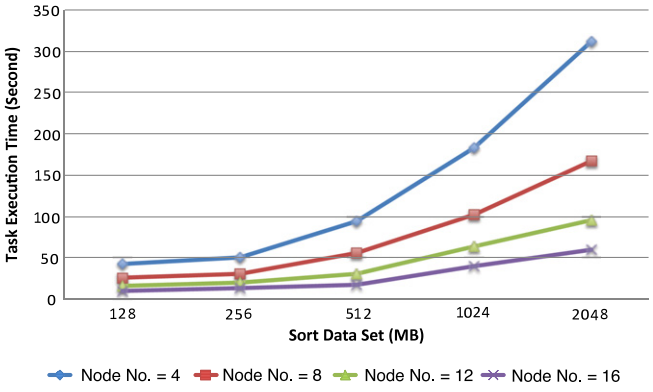


Fig. 16. Task execution time of Sort benchmark with various input data set.

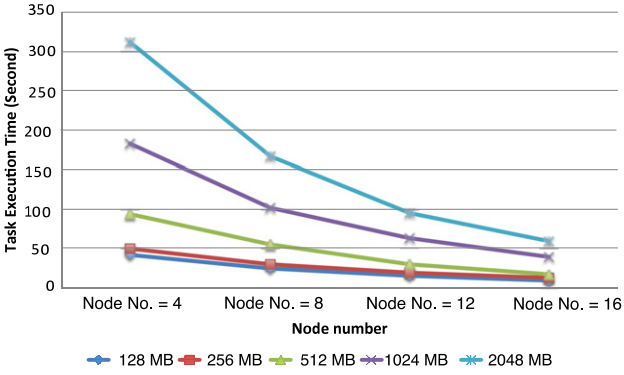


Fig. 17. Task execution time of Sort benchmark with G-Hadoop clusters.

sort) increases. Fig. 17 shows the Sort benchmark task execution time with G-Hadoop on the test bed with cluster node number. Fig. 17 indicates that G-Hadoop has a similar behavior with Test 1 in Section 5.1 when cluster size scales.

6. 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 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. We have run several experiments on our prototype. The results show that G-Hadoop has a comparable performance to the Apache Hadoop clusters and

scales nearly linearly with respect to the number of nodes in the cluster.

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

In future we would like to overcome the centralized network design of G-Hadoop master node and tracker through implementation of scalable peer-to-peer message routing and information dissemination structure. This is to help us in avoiding the problems of a single point of failure of the centralized network design approach. In particular, we will implement the distributed hash table based decentralized resource discovery technique, which will have the capability to handle complex cloud resource and G-Hadoop services' performance status (e.g., utilization, throughput, latency, etc.). We will also extensively leverage our past work [46] on grid resource provisioning for coordinating the scheduling decision making among distributed G-Hadoop services.

Acknowledgments

LW's work is funded by the "One-Hundred Talents Program" of the Chinese Academy of Sciences. DC's work is supported in part by the National Natural Science Foundation of China (grant No. 61272314), the Natural Science Foundation of Hubei Province of China (grant No. 2011CDB159), the Program for New Century Excellent Talents in University (NCET-11-0722), the Specialized Research Fund for the Doctoral Program of Higher Education (grant No. 20110145110010), and the Fundamental Research Funds for the Central Universities (CUG, Wuhan).

References

[1] F. Berman, Got data?: a guide to data preservation in the information age, Communications of the ACM 51 (2008) 50–56.
[2] Data intensive computing, Website. http://en.wikipedia.org/wiki/Data_Intensive_Computing.
[3] L. Wang, M. Kunze, J. Tao, G. von Laszewski, Towards building a cloud for scientific applications, Advances in Engineering Software 42 (9) (2011) 714–722.
[4] G. Brumfiel, High-energy physics: down the petabyte highway, Nature (7330) (2011) 282–283.
[5] L. Wang, C. Fu, Research advances in modern cyberinfrastructure, New Generation Computing 28 (2) (2010) 111–112.
[6] Condor dagman, Website. http://www.cs.wisc.edu/condor/dagman/.
[7] 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, D.S. Katz, Pegasus: a framework for mapping complex scientific workflows onto distributed systems, Science Programming 13 (2005) 219–237.
[8] Y. Zhao, M. Hategan, B. Clifford, I.T. Foster, G. von Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, M. Wilde, Swift: fast, reliable, loosely coupled parallel computation, in: IEEE SCW, IEEE Computer Society, Salt Lake City, Utah, USA, 2007, pp. 199–206.
[9] I. Altintas, B. Ludaescher, S. Klasky, 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, ACM, New York, NY, USA, 2006.
[10] L. Wang, D. Chen, F. Huang, Virtual workflow system for distributed collaborative scientific applications on grids, Computers & Electrical Engineering 37 (3) (2011) 300–310.
[11] L. Wang, M. Kunze, J. Tao, Performance evaluation of virtual machine-based grid workflow system, Concurrency and Computation: Practice and Experience 20 (15) (2008) 1759–1771.
[12] L. Wang, G. von Laszewski, J. Tao, M. Kunze, Virtual data system on distributed virtual machines in computational grids, International Journal of Ad Hoc and Ubiquitous Computing 6 (4) (2010) 194–204.
[13] P. Missier, S. Soiland-Reyes, S. Owen, W. Tan, A. Nenadic, I. Dunlop, A. Williams, T. Oinn, C. Goble, Taverna, reloaded, in: Proceedings of the 22nd International Conference on Scientific and Statistical Database Management, SSDBM'10, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 471–481.
[14] L. Wang, G. von Laszewski, A.J. Younge, X. He, M. Kunze, J. Tao, C. Fu, Cloud computing: a perspective study, New Generation Computing 28 (2) (2010) 137–146.
[15] R. Ranjan, L. Zhao, X. Wu, A. Liu, A. Quiroz, M. Parashar, Peer-to-peer cloud provisioning: service discovery and load-balancing, Cloud Computing (2010) 195–217.
[16] R. Buyya, R. Ranjan, Special section: federated resource management in grid and cloud computing systems, Future Generation Computer Systems 26 (8) (2010) 1189–1191.
[17] D. Chen, L. Wang, G. Ouyang, X. Li, Massively parallel neural signal processing on a many-core platform, Computing in Science and Engineering (2011).
[18] X. Yang, L. Wang, G. von Laszewski, Recent research advances in e-science, Cluster Computing 12 (4) (2009) 353–356.
[19] H. Zhu, T.K.Y. Chan, L. Wang, R.C. Jegathese, A distributed 3d rendering application for massive data sets, IEICE Transactions 87-D (7) (2004) 1805–1812.
[20] C. Moretti, J. Bulosan, D. Thain, P.J. Flynn, All-pairs: an abstraction for data-intensive cloud computing, in: IPDPS, April 2008, pp. 1–11.
[21] R. Grossman, Y. Gu, Data mining using high performance data clouds: experimental studies using sector and sphere, in: Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'08, ACM, New York, NY, USA, 2008, pp. 920–927.
[22] Y. Yu, M. Isard, D. Fetterly, M. Budiu, U. Erlingsson, P.K. Gunda, J. Currey, Dryadlinq: a system for general-purpose distributed data-parallel computing using a high-level language, in: Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, OSDI'08, USENIX Association, Berkeley, CA, USA, 2008, pp. 1–14.
[23] D. Logothetis, K. Yocum, Wide-scale data stream management, in: USENIX 2008 Annual Technical Conference on Annual Technical Conference, USENIX Association, Berkeley, CA, USA, 2008, pp. 405–418.
[24] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Communications of the ACM 51 (2008) 107–113.
[25] Apache Hadoop project, Web Page. http://hadoop.apache.org/.
[26] B. He, W. Fang, Q. Luo, N.K. Govindaraju, T. Wang, Mars: a mapreduce framework on graphics processors, in: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, PACT'08, ACM, New York, NY, USA, 2008, pp. 260–269.
[27] R. Chen, H. Chen, B. Zang, Tiled-mapreduce: optimizing resource usages of data-parallel applications on multicore with tiling, in: Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques, PACT'10, ACM, New York, NY, USA, 2010, pp. 523–534.
[28] Y. Shan, B. Wang, J. Yan, Y. Wang, N. Xu, H. Yang, Fpmr: mapreduce framework on fpga, in: Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA'10, ACM, New York, NY, USA, 2010, pp. 93–102.
[29] C. Ranger, R. Raghuraman, A. Penmetsa, G.R. Bradski, C. Kozyrakis, Evaluating mapreduce for multi-core and multiprocessor systems, in: 13st International Conference on High-Performance Computer Architecture, 2007, pp. 13–24.
[30] R.M. Yoo, A. Romano, C. Kozyrakis, Phoenix rebirth: scalable mapreduce on a large-scale shared-memory system, in: Proceedings of the 2009 IEEE International Symposium on Workload Characterization, IEEE, Austin, TX, USA, 2009, pp. 198–207.
[31] M.M. Rafique, B. Rose, A.R. Butt, D.S. Nikolopoulos, Supporting mapreduce on large-scale asymmetric multi-core clusters, SIGOPS Operating Systems Review 43 (2009) 25–34.
[32] S. Ibrahim, H. Jin, B. Cheng, H. Cao, S. Wu, L. Qi, Cloudlet: towards mapreduce implementation on virtual machines, in: Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing, HPDC'09, ACM, New York, NY, USA, 2009, pp. 65–66.
[33] C. Jin, R. Buyya, Mapreduce programming model for.net-based cloud computing, in: Euro-Par, in: Lecture Notes in Computer Science, vol. 5704, Springer, 2009, pp. 417–428.
[34] S. Pallickara, J. Ekanayake, G. Fox, Granules: a lightweight, streaming runtime for cloud computing with support for map-reduce, in: CLUSTER, IEEE, New Orleans, Louisiana, USA, 2009, pp. 1–10.
[35] C. Miceli, M. Miceli, S. Jha, H. Kaiser, A. Merzky, Programming abstractions for data intensive computing on clouds and grids, in: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID'09, IEEE Computer Society, Washington, DC, USA, 2009, pp. 478–483.
[36] H. Lin, X. Ma, J. Archuleta, W.-C. Feng, M. Gardner, Z. Zhang, Moon: mapreduce on opportunistic environments, in: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC'10, ACM, New York, NY, USA, 2010, pp. 95–106.
[37] A. Dou, V. Kalogeraki, D. Gunopulos, T. Mielikainen, V.H. Tuulos, Misco: a mapreduce framework for mobile systems, in: Proceedings of the 3rd International Conference on PErvasive Technologies Related to Assistive Environments, PETRA'10, ACM, New York, NY, USA, 2010, pp. 32:1–32:8.
[38] Apache Hadoop on Demand (HOD), Website. http://hadoop.apache.org/common/docs/r0.21.0/hod_scheduler.html.
[39] S. Krishnan, M. Tatineni, C. Baru, myhadoop—hadoop-on-demand on traditional hpc resources, University of California, San Diego, Technical Report, 2011.
[40] O. Tatebe, K. Hiraga, N. Soda, Gfarm grid file system, New Generation Computing 28 (3) (2010) 257–275.
[41] Torque resource manager, Website. http://www.clusterresources.com/products/torque-resource-manager.php.
[42] Distributed Resource Management Application API (DRMAA), Website. http://drmaa.org/.
[43] W. Sun, J. Tao, R. Ranjan, A. Streit, A security framework in G-Hadoop for data-intensive computing applications across distributed clusters, Journal of Computer and System Sciences.
[44] D.H. Bailey, The bbp algorithm for pi, Sep. 2006. http://crd.lbl.gov/dhbailey/dhbpapers/bbp-alg.pdf.

[45] Hadoop sort benchmark, Website. http://wiki.apache.org/hadoop/Sort.
[46] R. Ranjan, A. Harwood, R. Buyya, Coordinated load management in peer-to-peer coupled federated grid systems, Journal of Supercomputing 61 (2) (2012) 292–316. http://dx.doi.org/10.1007/s11227-010-0426-y. [Online], Available.

Dr. Lizhe Wang currently is Professor at Earth Observation and Digital Earth, Chinese Academy of Sciences. He also holds a "Chutian Chair" professor at the China University of Geosciences. Dr. Wang's research interests include data-intensive computing, high performance computing, and Grid/Cloud computing.

Dr. Jie Tao is a researcher at Steinbuch Centre for Computing (SCC), Karlsruhe Institute of Technology (KIT), Germany. Her research interests include parallel computing, Grid computing and Cloud computing.

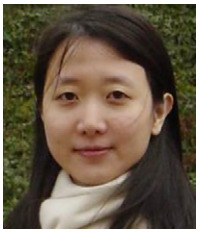
Dr. Rajiv Ranjan is a Research Scientist and Project Leader in the CSIRO ICT Center, Information Engineering Laboratory, Canberra, where he is working on projects related to cloud and service computing. Dr. Ranjan is broadly interested in the emerging areas of cloud, grid, and service computing. Though a recent graduate, his *h*-index is 14, with a total citation count of 650+ (source: Google Scholar citations-gadget). Dr. Ranjan has often served as Guest Editor for leading distributed systems and software engineering journals including Future Generation Computer Systems (Elsevier Press), Concurrency and Computation: Practice and Experience (John Wiley & Sons), and Software: Practice and Experience (Wiley InterScience). He was the Program Chair for 2010–12 Australasian Symposium on Parallel and Distributed Computing and 2010 IEEE TCSC Doctoral Symposium. He serves as the editor of IEEE TCSC Newsletter. He has also recently initiated (as chair) the IEEE TCSC Technical area on Cloud Computing. He has authored/co-authored 36 publications including 5 book chapters, 4 books, 13 iournal papers, and 14 conference papers.

A journal paper that appeared in the IEEE Communications Surveys and Tutorial Journal (impact factor 3.692 and the 5-year impact factor is 8.462) was named "Outstanding Paper on New Communications Topics for 2009" by IEEE Communications Society, USA. IEEE Communications Surveys and Tutorial is ranked as no # 1 journal (among other journals including IEEE/ACM Transactions on Networking, IEEE Communications Letters, IEEE Transactions on Wireless Communications, IEEE Transactions on Communications, etc.) in the field of communications based on the last five years' impact factor.

Though his Ph.D. was awarded only two years ago, already the quality of work undertaken as part of that degree and his subsequent post-doctoral research undertaken at the University of Melbourne and UNSW has been recognised, often cited, and adopted by both industry and academia. For example, the Alchemi toolkit (http://www.cloudbus.org/alchemi/projects.html) is used by CSIRO Physics, Biology, and Natural Sciences researchers as a platform for solving critical problems; by e-Water CRC to create environment simulation models for natural resource modelling; and by the Friedrich Miescher Institute (FMI), Switzerland; Tier Technologies, USA; Satyam Computers, India; and Correlation Systems Ltd., Israel. CloudSim (http://www.cloudbus.org/cloudsim/) is used extensively by Hewlett-Packard Labs, Texas A&M University, and Duke University in the USA; and Tsingua University in China to study and evaluate performance measurements of Cloud resources and application management techniques.

Dr. Holger Marten is a researcher at Steinbuch Centre for Computing (SCC), Karlsruhe Institute of Technology (KIT), Germany. His research interests include Grid computing and Cloud computing.

Dr. Achim Streit is the Director of Steinbuch Centre for Computing (SCC), Karlsruhe Institute of Technology (KIT), and a Professor for Distributed and Parallel High Performance Systems, Institute of Telematics, Department of Informatics, Karlsruhe Institute of Technology (KIT), Germany. His research includes high performance computing, Grid computing and Cloud computing.



Dr. Jingying Chen is a professor at Central Noraml University of China. Her research includes pattern recognition, artificial intelligence, and distributed computing.

Dr. Dan Chen is a professor at School of Computer, Chinese University of Geosciences. His research interests includes GPGPU, neural signal processing and High Performance computing.