Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters

Jiong Xie, Shu Yin, Xiaojun Ruan, Zhiyang Ding, Yun Tian,
James Majors, Adam Manzanares, and Xiao Qin

Department of Computer Science and Software Engineering
Auburn University, Auburn, AL 36849-5347
Email: {jzx0009, szy0004, xzr0001, dingzhi, tianyun, majorjh, acm0008}@eng.auburn.edu,
xqin@auburn.edu http://www.eng.auburn.edu/∼xqin

***Abstract*—MapReduce has become an important distributed processing model for large-scale data-intensive applications like data mining and web indexing. Hadoop–an open-source implementation of MapReduce is widely used for short jobs requiring low response time. The current Hadoop implementation assumes that computing nodes in a cluster are homogeneous in nature. Data locality has not been taken into account for launching speculative map tasks, because it is assumed that most maps are data-local. Unfortunately, both the homogeneity and data locality assumptions are not satisfied in virtualized data centers. We show that ignoring the data-locality issue in heterogeneous environments can noticeably reduce the MapReduce performance. In this paper, we address the problem of how to place data across nodes in a way that each node has a balanced data processing load. Given a data-intensive application running on a Hadoop MapReduce cluster, our data placement scheme adaptively balances the amount of data stored in each node to achieve improved data-processing performance. Experimental results on two real data-intensive applications show that our data placement strategy can always improve the MapReduce performance by rebalancing data across nodes before performing a data-intensive application in a heterogeneous Hadoop cluster.**

I. Introduction

An increasing number of popular applications become data-intensive in nature. In the past decade, the World Wide Web has been adopted as an ideal platform for developing data-intensive applications, since the communication paradigm of the Web is sufficiently open and powerful. Representative data-intensive Web applications include search engines, online auctions, webmail, and online retail sales. Data-intensive applications like data mining and web indexing need to access ever-expanding data sets ranging from a few gigabytes to several terabytes or even petabytes. Google, for example, leverages the MapReduce model to process approximately twenty petabytes of data per day in a parallel fashion [8]. MapReduce is an attractive model for parallel data processing in high-performance cluster computing environments. The scalability of MapReduce is proven to be high, because a job in the MapReduce model is partitioned into numerous small tasks running on multiple machines in a large-scale cluster.

Hadoop – a popular open-source implementation of the Google's MapReduce model is primarily developed by Yahoo [1]. Hadoop is used by Yahoo servers, where hundreds of terabytes of data are generated on at least 10,000 cores [4]. Facebook makes use of Hadoop to process more than 15 terabytes of new data per day. In addition to Yahoo and Facebook, a wide variety of websites like Amazon and Last.fm are employing Hadoop to manage massive amount of data on a daily basis [14]. Apart from Web data-intensive applications, scientific data-intensive applications (e.g., seismic simulation and natural language processing) take maximum benefits from the Hadoop system [6][14].

The MapReduce framework can simplify the complexity of running distributed data processing functions across multiple nodes in a cluster, because MapReduce allows a programmer with no specific knowledge of distributed programming to create his/her MapReduce functions running in parallel across multiple nodes in the cluster. MapReduce automatically handles the gathering of results across the multiple nodes and return a single result or set. More importantly, the MapReduce platform can offer fault tolerance that is entirely transparent to programmers [8].

We observe that data locality is an determining factor for the MapReduce performance. To balance load, Hadoop distributes data to multiple nodes based on disk space availability. Such data placement strategy is very practical and efficient for a homogeneous environment where nodes are identical in terms of both computing and disk capacity. In homogeneous computing environments, all the nodes have identical workload, indicating that no data needs to be moved from one node into another. In a heterogeneous cluster, however, a high-performance nodes can complete processing local data faster than a low-performance node. After the fast node finished processing data residing in its local disk, the node has to handle unprocessed data in a remote slow node. The overhead of transferring unprocessed data from slow nodes to fast peers is high if the amount of moved data is huge. An approach to improve MapReduce performance in heterogeneous computing environments is to significantly reduce the amount of data moved between slow and fast nodes in a heterogeneous cluster. To balance data load in a heterogeneous Hadoop cluster, we are motivated to investigate data placement schemes, which aim to partition a large data set into data fragments that are distributed across multiple heterogeneous nodes in a cluster.

In this study, we developed a data placement mechanism in the Hadoop distributed file system or HDFS to initially distribute a large data set to multiple nodes in accordance to the computing capacity of each node. More specifically, we implemented a data reorganization algorithm in addition to a data redistribution algorithm in HDFS. The data reorganization and redistribution algorithms implemented in HDFS can be used to solve the data skew problem due to dynamic data insertions and deletions.

The rest of the paper is organized as follows. Section II gives an overview of the MapReduce programming model and a brief introduction to the Hadoop distribution file system (HDFS). The data distribution algorithm is described in Section III. Section IV describes the implementation details of our data placement mechanism. In Section V, we present the evaluation results. Section VI reviews related work and Section VII concludes the paper with future research directions.

II. Background and Motivation

A. MapReduce Overview

The MapReduce programming model was proposed by Google to support data-intensive applications running on parallel computers like commodity clusters. Two important functional programming primitives in MapReduce are *Map* and *Reduce*. The Map function is applied on application-specific input data to generate a list of intermediate $< key, value >$ pairs. Then, the Reduce function is applied to the set of intermediate pairs with the same key. Typically, the Reduce function produces zero or more output pairs by performing a merging operation. All the output pairs are finally sorted based on their key values. Programmers only need to implement the Map and Reduce functions, because a MapReduce programming framework can facilitate some operations (e.g., grouping and sorting) on a set of $< key, value >$ pairs.

The beauty of the MapReduce model lies in its simplicity, because the programmers just have to focus on data-processing functionality rather than on parallelism details. The programmers provide high-level parallelism information, thereby allowing the Map and Reduce functions to be executed in parallel across multiple nodes.

In the past few years, the MapReduce framework has been employed to develop a wide variety of data-intensive applications (e.g., data mining and bioinformatics) in large-scale systems. There exists several implementations of MapReduce on various hardware platforms. For example, Phoenix is a MapReduce implementation on multi-core processors [13]. Mars is an efficient implementation of the MapReduce model on graphics processors or GPUs [5]. Mpa-Reduce-Merge is a MapReduce implementation for relational databases [9].

B. Hadoop and Hadoop Distributed File System

Hadoop is a successful implementation of the MapReduce model. The Hadoop framework consists of two main components: the MapReduce language and the Hadoop's Distributed File System (or HDFS for short). The Hadoop runtime system coupled with HDFS manages the details of parallelism and concurrency to provide ease of parallel programming with reinforced reliability. In a Hadoop cluster, a master node controls a group of slave nodes on which the Map and Reduce functions run in parallel. The master node assigns a task to a slave node that has any empty task slot.

Typically, computing nodes and storage nodes in a Hadoop cluster are identical from the hardware's perspective. In other words, the Hadoop's Map/Reduce framework and the Hadoop's HDFS are, in many cases, running on a set of homogeneous nodes including both computing and storage nodes. Such a homogeneous configuration of Hadoop allows the Map/Reduce framework to effectively schedule computing tasks on an array of storage nodes where data files are residing, leading to a high aggregate bandwidth across the entire Hadoop cluster.

An input file passed to Map functions resides on the Hadoop distributed file system on a cluster. Hadoop's HDFS splits the input file into even-sized fragments, which are distributed to a pool of slaves for further MapReduce processing. HDFS closely resembles the Google file system or GFS [15]. Unlike other distributed file systems, HDFS aims to provide high throughput access to large application data sets. HDFS is highly reliable because each file fragment stored on a data node is replicated for fault-tolerance purpose. HDFS uses three-way replication to ensure that files residing in a Hadoop cluster are always intact in three separate nodes in the cluster.

C. Motivation

Fig. 1 shows how a Hadoop program accesses HDFS in a cluster. The MapReduce program directs file queries to a namenode, which in turn passes the file requests to corresponding nodes in the cluster. Then, the data nodes concurrently feed Map functions in the MapReduce program with large amount data. When new application data are written to a file in HDFS, file fragments of the file are stored on multiple data nodes across the Hadoop cluster. HDFS distributes file fragments across the cluster, assuming that all the nodes have identical computing capacity. Such a homogeneity assumption, which can potentially hurt the Hadoop performance of heterogeneous clusters, motivates

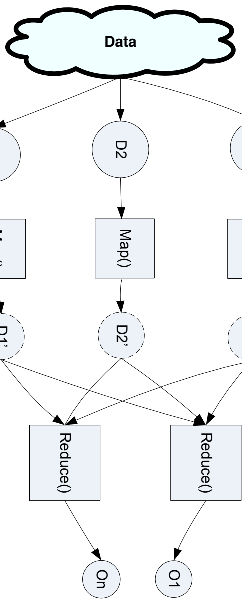


Figure 1: The MapReduce programming model, where a Hadoop application accesses Hadoop distributed file system (HDFS) in a cluster.

us to develop data placement schemes that can noticeably improve the performance of heterogeneous Hadoop clusters.

III. Data Placement Management

A. Data Placement in Heterogeneous Clusters

In a cluster where each node has a local disk, it is efficient to move data processing operations to nodes where application data are located. If data are not locally available in a processing node, data have to be migrated via network interconnects to the node that performs the data processing operations. Migrating huge amount of data leads to excessive network congestion, which in turn can deteriorate system performance. HDFS enables Hadoop MapReduce applications to transfer processing operations toward nodes storing application data to be processed by the operations.

In a heterogeneous cluster, the computing capacities of nodes may vary significantly. A high-speed node can finish processing data stored in a local disk of the node faster than low-speed counterparts. After a fast node complete the processing of its local input data, the node must support load sharing by handling unprocessed data located in one or more remote slow nodes. When the amount of transferred data due to load sharing is very large, the overhead of moving unprocessed data from slow nodes to fast nodes becomes a critical issue affecting Hadoop's performance. To boost the performance of Hadoop in heterogeneous clusters, we aim to minimize data movement between slow and fast nodes. This goal can be achieved by a data placement scheme that distribute and store data across multiple heterogeneous nodes based on their computing capacities. Data movement can be reduced if the number of file fragments placed on the disk of each node is proportional to the node's data processing speed.

To achieve the best I/O performance, one may make replicas of an input data file of a Hadoop application in a way that each node in a Hadoop cluster has a local copy of the input data. Such a data replication scheme can, of course, minimize data transfer among slow and fast nodes in the cluster during the execution of the Hadoop application. The data-replication approach has several limitations. First, it is very expensive to create replicas in a large-scale cluster. Second, distributing a large number of replicas can wasterfully consume scarce network bandwidth in Hadoop clusters. Third, storing replicas requires an unreasonably large amount of disk capacity, which in turn increases the cost of Hadoop clusters.

Although all replicas can be produced before the execution of Hadoop applications, significant efforts must be make to reduce the overhead of generating replicas. If the data-replication approach is employed in Hadoop, one has to address the problem of high overhead for creating file replicas by implementing a low-overhead file-replication mechanism. For example, Shen and Zhu developed a proactive low-overhead file replication scheme for structured peer-to-peer networks [16]. Shen and Zhu's scheme may be incorporated to overcome this limitation.

To address the above limitations of the data-replication approach, we are focusing on data-placement strategies where files are partitioned and distributed across multiple nodes in a Hadoop cluster without being duplicated. Our data placement approach does not require any comprehensive scheme to deal with data replicas.

In our data placement management mechanism, two algorithms are implemented and incorporated into Hadoop's HDFS. The first algorithm is to initially distribute file fragments to heterogeneous nodes in a cluster (see Section III-B). When all file fragments of an input file required by computing nodes are available in a node, these file fragments are distributed to the computing nodes. The second data-placement algorithm is used to reorganize file fragments to solve the data skew problem (see Section III-C). There two cases in which file fragments must be reorganized. First, new computing nodes are added to an existing cluster to have the cluster expanded. Second, new data is appended to an existing input file. In both cases, file fragments distributed by the initial data placement algorithm can be disrupted.

B. Initial Data Placement

The initial data placement algorithm begins by first dividing a large input file into a number of even-sized fragments. Then, the data placement algorithm assigns fragments to nodes in a cluster in accordance to the nodes' data processing speed. Compared with low-performance nodes, high-performance nodes are excepted to store and process more file fragments. Let us consider a MapReduce application and its input file in a heterogeneous Hadoop cluster. Regardless of the heterogeneity in node processing power, the intial data placement scheme has to distribute the fragments of the input file so that all the nodes can complete processing their local data within almost the same time.

In our experiments we observed that the computing capability of each node is quite stable for certain tested Hadoop applications, because the response time of these Hadoop applications on each node is linearly proportional to input data size. As such, we can quantify each node's processing speed in a heterogeneous cluster using a new term called computing ratio. The computing ratio of a computing node with respect to a Hadoop application can be calculated by profiling the application (see Section IV-A for details on how to determine computing ratios). It is worth noting that the computing ratio of a node may vary from application to application.

C. Data Redistribution

Input file fragments distributed by the initial data placement algorithm might be disrupted due to the following reasons: (1) new data is appended to an existing input file; (2) data blocks are deleted from the existing input file; and (3) new data computing nodes are added into an existing cluster. To address this dynamic data load-balancing problem, we implemented a data redistribution algorithm to reorganize file fragments based on computing ratios.

The data redistribution procedure is described as the following steps. First, like initial data placement, information regarding the network topology and disk space utilization of a cluster is collected by the data distribution server. Second, the server creates two node lists: a list of nodes in which the number of local fragments in each node exceeds its computing capacity and a list of nodes that can handle more local fragments because of their high performance. The first list is called over-utilized node list; the second list is termed as under-utilized node list. Third, the data distribution server repeatedly moves file fragments from an over-utilized node to an underutilized node until the data load are evenly distributed. In a process of migrating data between a pair of an over-utilized and an underutilized nodes, the server moves file fragments from a source node in the over-utilized node list to a destination node in the underutilized node list. Note that the server decides the number of bytes rather than fragments and moves fragments from the source to the destination node. The above data migration process is repeated until the number of local fragments in each node matches its speed measured by computing ratio.

IV. Implementation Details

A. Measuring Heterogeneity

Before implementing the initial data placement algorithm, we need to quantify the heterogeneity of a Hadoop cluster in terms of data processing speed. Such processing speed highly depends on data-intensive applications. Thus, heterogeneity measurements in the cluster may change while executing different MapReduce applications. We introduce a metric - called computing ratio - to measure each node's processing speed in a heterogeneous cluster. Computing ratios are determined by a profiling procedure carried out in the following steps. First, the data processing operations of a given MapReduce application are separately performing in each node. To fairly compare processing speed, we ensure that all the nodes process the same amount of data. For example, in one of our experiments the input file size is set to 1GB. Second, we record the response time of each node performing the data processing operations. Third, the shortest response time is used as a reference to normalize the response time measurements. Last, the normalized values, called computing ratios, are employed by the data placement algorithm to allocate input file fragments for the given MapReduce application.

Now let us consider an example to demonstrate how to calculate computing ratios used to guide the data distribution process. Suppose there are three heterogeneous nodes (i.e., Node A, B and C) in a Hadoop cluster. After running a Hadoop application on each node, one collects the response time of the application on node A, B and C is 10, 20 and 30 seconds, respectively. The response time of the application on node C is the shortest. Therefore, the computing ratio of node A with respect to this application is set to 1, which becomes a reference used to determine computing ratios of node B and C. Thus, the computing ratios of node B and C are 2 and 3, respectively. Recall that the computing capacity of each node is quite stable with respect to a Hadoop application. Hence, the computing ratios are independent of input file sizes. Table I shows the response times and computing ratios for each node in a Hadoop cluster. Table I also shows the number of file fragments to be distributed to each node in the cluster. Intuitively, the fast computing node (i.e., node A) has to handle 30 file fragments whereas the slow node (i.e., 3) only needs to process 10 fragments.

Table I: Computing ratios, response times and number of file fragments for three nodes in a Hadoop cluster

Node	Responce time	Ratio	File fragments	Speed
Node A	10	1	30	Fastest
Node B	20	2	20	Average
Node C	30	3	10	Slowest

B. Sharing Files among Multiple Applications

The heterogeneity measurement of a cluster depends on data-intensive applications. If multiple MapReduce applications must process the same input file, the data placement mechanism may need to distribute the input file's fragments in several ways - one for each MapReduce application. In the case where multiple applications are similar in terms of data processing speed, one data placement decision may fit the needs of all the applications.

C. Data Distribution.

File fragment distribution is governed by a data distribution server, which constructs a network topology and calculates disk space utilization. For each MapReduce application, the server generates and maintains a node list containing computing-ratio information. The data distribution server applies the round-robin algorithm to assign input file fragments to heterogeneous nodes based on their computing ratios.

A small value of computing ratio of a node indicates a high speed of the node, meaning that the fast node must process a large number of file fragments. For example, let us consider a file comprised of 60 file fragments to be distributed to node A, B, and C. We assume the computing ratios of these three nodes are 1, 2 and 3, respectively (see Table I). Given the computing ratios, we can conclude that among the three computing nodes, node A is the fastest one whereas node B is the slowest node. As such, the number of file fragments assigned to each node is proportional to the node's processing speed. In this example, the data distribution server assigns 30 fragments to node A, 20 fragments to node B, and 10 fragments to node C (see Table I).

V. Evaluation

We used two data-intensive applications - Grep and WordCount - to evaluate the performance of our data placement mechanism in a heterogeneous Hadoop cluster. The tested cluster consists of five heterogeneous nodes, whose parameters are summarized in Table II. Both Grep and WordCount are two MapReduce applications running on Hadoop clusters. Grep is a tool searching for a regular expression in a text file; whereas WordCount is a program used to count words in text files.

Table II: Five Nodes in a Hadoop Heterogeneous Cluster

Node	CPU Model	CPU(hz)	L1 Cache(KB)
Node A	Intel Core 2 Duo	2 ×1G=2G	204
Node B	Intel Celeron	2.8G	256
Node C	Intel Pentium 3	1.2G	256
Node D	Intel Pentium 3	1.2G	256
Node E	Intel Pentium 3	1.2G	256

We followed the approach described in Section IV-A to obtain computing ratios of the five computing nodes with respect of the Grep and WordCount applications (see Table III). The computing ratios shown in Table III represent the heterogeneity of the Hadoop cluster with respect to Grep and WordCount. We conclude from the results given in Table III) that computing ratios of a Hadoop cluster are application dependent. For example, node A is 3.3 times faster than nodes C-E with respect to the Grep application; node A is 5 (rather than 3.3) times faster than nodes C-E when it comes to the WordCount application. The implication of the results is that given a heterogeneous cluster, one has to determine computing ratios for each Hadoop application. Note that computing ratios of each application only needs to be calculated once for each cluster. If the configuration of a cluster is updated, computing ratios must be determined again.

Table III: Computing Ratios of the Five Nodes with Respective of the Grep and WordCount Applications

Computer Node	Ratios for Grep	Ratios for WordCount
Node A	1	1
Node B	2	2
Node C	3.3	5
Node D	3.3	5
Node E	3.3	5

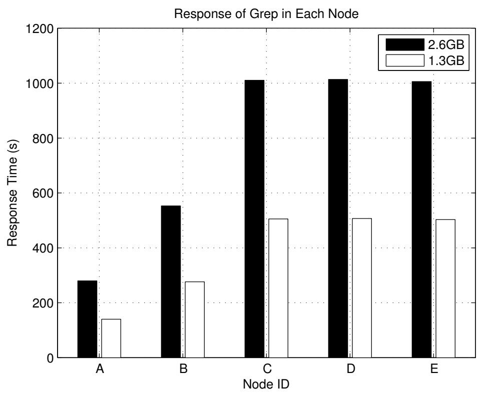
Figs. 2(a) and 2(b) show the response times of the Grep and WordCount application running on each node of the Hadoop cluster when the input file size is 1.3 GB and 2.6 GB, respectively. The results plotted in Figs. 2(a) and 2(b) suggest that computing ratios are independent of input file size, because the response times of Grep and WordCount are proportional to the file size. Regardless of input file size, the computation ratios for Grep and WordCount on the 5-node Hadoop clusters remain unchanged as listed in Table III.

Given the same input file size, Grep's response times are shorter than those of WordCount (see Figs. 2(a) and 2(b)). As a result, the computing ratios of Grep are different from those of WordCount (see Table III).

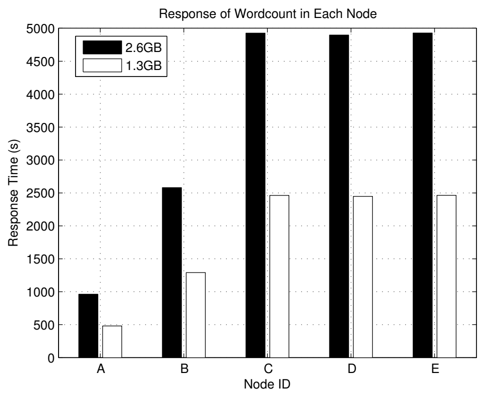
Table IV: Six Data Placement Decisions

Notation	Data Placement Decisions
S1-2-3.3	Dirstribuating files under the computing ratios of the grep. (This is an optimal data placement for Grep)
S1-2-5	Dirstribuating files under the computing ratios of the wordcount. (This is an optimal data placement for WordCount)
480 in each	Average distribution of files to each node.
All-in-A	Allocating all the files to node A.
All-in-B	Allocating all the files to node B.
All-in-C	Allocating all the files to node C.

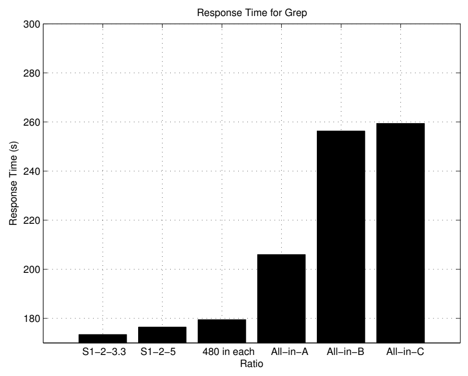
Now we are positioned to evaluate the impacts of data placement decisions on the response times of Grep and WordCount (see Figs. 2(c) and 2(d)). Table IV shows six representative data placement decisions, including two optimal data-placement decisions (see S1-2-3.3 and S1-2-5 in Table IV) for the Grep and WordCount applications.



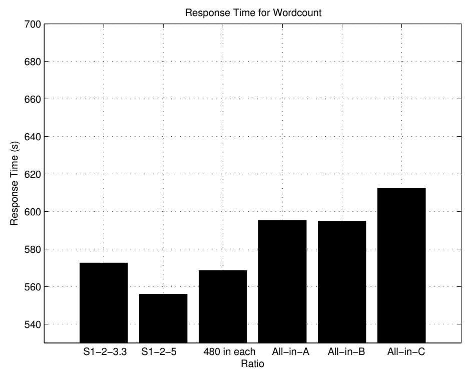
(a) Response time of Grep on each node.



(b) Response time of WordCount on each node.



(c) Impact of data placement on performance of Grep



(d) Impact of data placement on performance of WordCount

Figure 2: Response time of Grep and WordCount running on the 5-node Hadoop heterogeneous cluster.

The file fragments of input data are distributed and placed on the five heterogeneous nodes based on six different data placement decisions, among which two optimal decisions (i.e., S1-2-3.3 and S1-2-5 in Table IV) are made based on the computing ratios given Table III.

Let us use an example to show how the data distribution server relies on the S1-2-3.3 decision - optimal decision for Grep - in Table IV to distribute data to the five nodes of the tested cluster. Recall that the computing ratios of Grep on the 5-node Hadoop cluster are 1, 2, 3.3, 3.3, and 3.3 for nodes A-E (see Table III). We suppose there are 24 fragments of the input file for Grep. Thus, the data distribution server allocates 10 fragments to node A, 5 fragments to node B, and 3 fragments to nodes C-E.

Fig. 2(c) reveals the impacts of data placement on the response times of the Grep application. The first (leftmost) bar in Fig. 2(c) shows the response time of the Grep application by distributing file fragments based on Grep's computing ratios. For comparison purpose, the other bars in Fig. 2(c) show the response time of Grep on the 5-node cluster with the other five data-placement decisions. For example, the third bar in Fig. 2(c) is the response time of Grep when all the input file fragments are evenly distributed across the five nodes in the cluster. We observe from Fig. 2(c) that the first data placement decision (denoted as S1-2-3.3) leads to the best performance of Grep, because the input file fragments are distributed strictly according to the nodes' computing ratios. If the file fragments are placed using the "All-in-C" data-placement decision, Grep performs extremely poorly. Grep's response time is unacceptably long under the "All-in-C" decision, because all the input file fragments are placed on node C - one of the slowest node in the cluster. Under the "All-in-C" data placement decision, the fast nodes (i.e., nodes A and B) have to pay extra overhead to copy a significant amount of data from node C before processing the input data locally. Compared with the "All-in-C" decision, the optimal data placement decision reduces the response time of Grep by more than 33.1%.

Fig. 2(d) depicts the impacts of data placement decisions on the response times of WordCount. The second bar in Fig. 2(d) demonstrates the response time of the WordCount application on the cluster under an optimal data placement decision. In this optimal data placement case, the input file fragments are distributed based on the computing ratios listed in Table III. To illustrate performance improvement achieved by our new data placement strategy, we plotted the other five bars in Fig. 2(d) to show the response time of WordCount when the other five data-placement decisions are made and applied. Results plotted in Fig. 2(d) indicate that the response time of WordCount under the optimal "S1-2-5" data placement decision is the shortest compared with all the other five data placement decisions. For example, compared with the "All-in-C" decision, the optimal decision made by our strategy reduces the response time of WordCount by 10.2%. The "S1-2-5" data placement decision is proved to be the best, because this data placement decision is made based on the heterogeneity measurements - computing ratios in Table III. Again, the "All-in-C" data placement decision leads to the worst performance of WordCount, because under the "All-in-C" decision the fast nodes have copy a significant amount of data from node C. Moving data from node C to other fast nodes introduces extra overhead.

In summary, the results reported in Figs. 2(c) and 2(d) show that our data placement scheme can improve the performance of Grep and Wordcount by up to 33.1% and 10.2% with averages of 17.3% and 7.1%.

VI. Related Work

Implementations of MapReduce. Some research has been directed at implementing and evaluating performance of the MapReduce model [5][13][8][10]. For example, Ranger implemented MapReduce for shared-memory systems [13]. Phoenix leads to scalable performance for both multi-core chips and conventional symmetric multiprocessors. Bingsheng *et al.* developed Mars - a MapReduce framework for graphics processors(GPUs) [5]. The goal of Mars is to hide the programming complexity of GPUs behind the simple MapReduce interface.

MapReduce Frameworks in Heterogeneous Environments. Increasing evidence shows that heterogeneity problems must be tackled in MapReduce frameworks [11][12]. Zaharia *et al.* implemented a new scheduler - LATE - in Hadoop to improve MapReduce performance by speculatively executing tasks that hurt response time the most [12]. Asymmetric multi-core processors (AMPs) address the I/O bottleneck issue, using double-buffering and asynchronous I/O to support MapReduce functions in clusters with asymmetric components [11]. Chao *et al.* classified MapReduce workloads into three categories based on CPU and I/O utilization [17]. They designed the Triple-Queue Scheduler in light of the dynamic MapReduce workload prediction mechanism called MR-Predict. Although the above techniques can improve MapReduce performance of heterogeneous clusters, they do not take into account data locality and data movement overhead.

Parallel File Systems. There are two types of file systems handling large files for clusters, namely, parallel file systems and Internet service file systems [18]. Representative parallel file systems in clusters are Lustre [3] and PVFS (Parallel Virtual File System) [2]. Hadoop distribution file system(HDFS) [7] is a popular Internet service file system that provides the right abstraction for data processing in Mapreduce frameworks.

VII. Conclusions and Future Work

We identified performance problem in HDFS (Hadoop Distributed File System) on heterogeneous clusters. Motivated by the performance degradation caused by heterogeneity, we have designed and implemented a data placement mechanism in HDFS. The new mechanism distributes fragments of an input file to heterogeneous nodes based on their computing capacities. Our approach improves performance of Hadoop heterogeneous clusters.

Our future research will foucus on handling the data redundance issue of data allocation in the cluster, and designing a dynamic data distribution mechanism for mutliple data intensive applications working together.

Availability

The source code of this project is freely available at: http://www.eng.auburn.edu/∼xqin/software/hdfs-hc/.

Acknowledgment

The work reported in this paper was supported by the US National Science Foundation under Grants CCF-0845257 (CAREER), CNS-0917137 (CSR), CNS-0757778 (CSR), CCF-0742187 (CPA), CNS-0831502 (CyberTrust), CNS-0855251 (CRI), OCI-0753305 (CI-TEAM), DUE-0837341 (CCLI), and DUE-0830831 (SFS), as well as Auburn University under a startup grant and a gift (Number 2005-04-070) from the Intel Corporation.

References

[1] `http://lucene.apache.org/hadoop`.

[2] Parallel virtual file system, version 2. `http://www.pvfs2.org`.

[3] A scalable, high performance file system. `http://lustre.org`.

[4] Yahoo! launches worlds largest hadoop production application. `http://tinyurl.com/2hgzv7`.

[5] B.He, W.Fang, Q.Luo, N.Govindaraju, and T.Wang. *Mars: a MapReduce framework on graphics processors*. ACM, 2008.

[6] C.Olston, B.Reed, U.Srivastava, R.Kumar, and A.Tomkins. Pig latin: a not-so-foreign language for data processing. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM, 2008.

[7] D.Borthakur. *The Hadoop Distributed File System: Architecture and Design*. The Apache Software Foundation, 2007.

[8] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI '04*, pages 137–150, 2008.

[9] H.Yang, A.Dasdan, R.Hsiao, and D.S.Parker. Map-reduce-merge: simplified relational data processing on large clusters. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1029–1040. ACM, 2007.

[10] M.Isard, M.Budiu, Y.Yu, A.Birrell, and D.Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pages 59–72. ACM, 2007.

[11] M.Rafique, B.Rose, A.Butt, and D.Nikolopoulos. Supporting mapreduce on large-scale asymmetric multi-core clusters. *SIGOPS Oper. Syst. Rev.*, 43(2):25–34, 2009.

[12] M.Zaharia, A.Konwinski, A.Joseph, Y.zatz, and I.Stoica. Improving mapreduce performance in heterogeneous environments. *In OSDI'08: 8th USENIX Symposium on Operating Systems Design and Implementation*, October 2008.

[13] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis. Evaluating mapreduce for multi-core and multiprocessor systems. *High-Performance Computer Architecture, International Symposium on*, 0:13–24, 2007.

[14] R.Pike, S.Dorward, R.Griesemer, and S.Quinlan. *Interpreting the data: Parallel analysis with Sawzall*, volume 13. IOS Press, 2005.

[15] S.Ghemawat, H.Gobioff, and S.Leung. The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43, 2003.

[16] Haiying Shen and Yingwu Zhu. A proactive low-overhead file replication scheme for structured p2p content delivery networks. *J. Parallel Distrib. Comput.*, 69(5):429–440, 2009.

[17] T.Chao, H.Zhou, Y.He, and L.Zha. *A Dynamic MapReduce Scheduler for Heterogeneous Workloads*. IEEE Computer Society, 2009.

[18] W.Tantisiriroj, S.Patil, and G.Gibson. Data-intensive file systems for internet services: A rose by any other name ... *Carnegie Mellon University Parallel Data Lab Technical Report CMU-PDL-08-114*, October 2008.

VIII. Biographies

Jiong Xie received the BS and MS degrees in computer science from BUAA (Beijing University of Aeronautics and Astronautics), China, in 2004 and 2008. He is currently working toward the PhD degree at the Department of Computer Science and Software Engineering, Auburn University. His research interests include scheduling techniques and parallel algorithms for clusters, and also multi-core processors and software techniques for I/O-intensive applications.

Shu Yin received the BS degree in communication engineering and the MS degree in signal and information processing from Wuhan University of Technology (WUT) in 2006 and 2008, respectively. He is currently working toward the PhD degree at the Department of Computer Science and Software Engineering, Auburn University. His research interests include storage systems, reliability modeling, fault tolerance, energy-efficient computing, and wireless communications. He is a student member of the IEEE.

Xiaojun Ruan received the BS degree in computer science from Shandong University in 2005. He is currently working toward the PhD degree at the Department of Computer Science and Software Engineering, Auburn University. His research interests are in parallel and distributed systems, storage systems, real-time computing, performance evaluation, and fault tolerance. His research interests focus on highperformance parallel cluster computing, storage system, and distributed system. He is a student member of the IEEE.

Zhiyang Ding received the BS degree in computer science from Tianjin University, Tianjin, China in 2006. From 2006, he works toward the PhD degree at the Department of Computer Science and Software Engineering, Auburn University. His research interests are Smart Disk system, MapReduce programming model, parallel computing and high-performance computing. He is a student member of the IEEE.

Yun Tian graduated from Northwest University, computer science and technology major in Information Science and Technology College with the BS degree in 2006, Xi'an, China. She is currently working toward the PhD degree at the Department of Computer Science and Software Engineering at Auburn University. Her research interests focus on security issues in storage system, distributed system, parallel computing and cloud computing.

James Majors received the BS degree in Software Engineering from Auburn University in 2009. He is currently working toward the PhD degree at the Department of Computer Science and Software Engineering, Auburn University. His research interests include storage systems, energy-efficient computing, operating systems, and filesystems. His research interests focus on secure filesystems and distributed systems. He is a student member of the IEEE.

Adam Manzanares received the BS degree in computer science from the New Mexico Institute of Mining and Technology, United States, in 2006. He is currently working toward the PhD degree at the Department of Computer Science and Software Engineering, Auburn University. During the Summers of 2002-2007, he has worked as a student intern at the Los Alamos National Laboratory. His research interests include energy-efficient computing, modeling and simulation, and high-performance computing. He is a student member of the IEEE.

Xiao Qin received the BS and MS degrees in computer science from Huazhong University of Science and Technology in 1992 and 1999, respectively, and the PhD degree in computer science from the University of Nebraska-Lincoln in 2004. Currently, he is an assistant professor in the Department of Computer Science and Software Engineering at Auburn University. Prior to joining Auburn University in 2007, he had been an assistant professor with New Mexico Institute of Mining and Technology (New Mexico Tech) for three years. He received the US National Science Foundation (NSF) CAREER Award in 2009. In 2007, he received an NSF CPA Award and an NSF CSR Award. His research interests include parallel and distributed systems, storage systems, fault tolerance, real-time systems, and performance evaluation. His research is supported by the US National Science Foundation, Auburn University, and Intel Corporation. He had served as a subject area editor of the IEEE Distributed System Online (2000-2001). He has been on the program committees of various international conferences, including IEEE Cluster, IEEE IPCCC, and ICPP. He is a senior member of the IEEE.