HybridMR: a New Approach for Hybrid MapReduce Combining Desktop Grid and Cloud Infrastructures

Bing Tang, Haiwu He, Gilles Fedak

CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. (2015)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3515

SPECIAL ISSUE PAPER

HybridMR: a new approach for hybrid MapReduce combining desktop grid and cloud infrastructures

Bing Tang[1,*,†], Haiwu He[2] and Gilles Fedak[3]

[1]*School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan, 411201, China*
[2]*Computer Network Information Center, Chinese Academy of Sciences, Beijing, 100864, China*
[3]*LIP Laboratory, University of Lyon, Lyon, 69364, France*

SUMMARY

This paper introduces HybridMR, a novel model for the execution of MapReduce (MR) computation on hybrid computing environment. Using this model, high performance cloud resources and heterogeneous desktop personal computers (PCs) in Internet or Intranet can be integrated to form a hybrid computing environment. Thanks to HybridMR, the computation and storage capability of large scale desktop PCs can be fully utilized to process large scale datasets. HybridMR relies on two innovative solutions to enable such large scale data-intensive computation. The first one is HybridDFS, which is a hybrid distributed file system. HybridDFS features reliable distributed storage that alleviates the volatility of desktop PCs, thanks to fault tolerance and file replication mechanism. The second innovation is a new node priority-based fair scheduling (NPBFS) algorithm has been developed in HybridMR to achieve both data storage balance and job assignment balance by assigning each node a priority through quantifying CPU speed, memory size, and input and output capacity. In this paper, we describe the HybridMR, HybridDFS, and NPBFS. We report on performance evaluation results, which show that the proposed HybridMR not only achieves reliable MR computation, reduces task response time, and improves the performance of MR, but also reduces the computation cost and achieves a greener computing mode. Copyright © 2015 John Wiley & Sons, Ltd.

Received 1 November 2014; Revised 8 January 2015; Accepted 16 January 2015

KEY WORDS: hybrid computing environment; volunteer computing; MapReduce; distributed file system; fault tolerance

1. INTRODUCTION

In the past decade, desktop grid and volunteer computing systems (DGVCSs) have been proved an effective solution to provide scientists with tens of teraflops from hundreds of thousands of resources. DGVCSs utilize free computing, network, and storage resources of idle desktop personal computers (PCs) distributed over Intranet or Internet environments for supporting large scale computation and storage. DGVCSs have been one of the largest and most powerful distributed computing systems in the world, offering a high return on investment for applications from a wide range of scientific domains, including computational biology, climate prediction, and high-energy physics [1–3].

On the other hand, cloud computing is emerging as a promising paradigm delivering information technology services as computing utilities, which are capable of providing a flexible, dynamic, resilient, and cost-effective infrastructure. A cloud instance (such as Amazon elastic compute cloud

*Correspondence to: Bing Tang, School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan, 411201, China.
†E-mail: btang@hnust.edu.cn

(EC2) instance) is a virtual server node, which can be used to run various applications. MapReduce (MR) is a programming model for data intensive application that was first introduced by Google in 2004 [4] and has attracted a lot of attentions recently. Hadoop is an open-source implementation of MR, which is widely used in Yahoo, Facebook, and Amazon. MR borrows ideas from functional programming, in order to simplify the parallel programming to process massive datasets, where programmers only define map and reduce tasks and do not need to care about complex processes. Now, many providers offer MR computing services using cloud infrastructure in the pay-as-you-go manner such as Amazon, Google, Rackspace, Aliyun, and so on.

Recently, there are some other MR implementations that are designed for large scale parallel data processing specialized on desktop grid or volunteer resources in Intranet or Internet such as BitDew-MapReduce [5], MOON [6], P2P-MapReduce [7], GiGi-MR [8], VMR [9], and so on. However, because there exists the correlation of volunteer or desktop failures, in order to achieve long-term and sustained high throughput, MR implementations adapted to volatile desktop environments cannot lack the support of high reliable cluster nodes.

To this end, this paper presents a hybrid computing environment, in which the cluster nodes and the volunteer computing nodes are integrated. For this hybrid computing environment, we propose and implement a MR parallel computation model that takes advantages of the computing capability of these two kinds of resource to execute reliable MR tasks.

The main challenges include three aspects: the first is how to deal with task failures caused by unreliable volunteer-computing node failures, the second is how to store the input data, the intermediate data, and the final results for MR applications, and the third is how to achieve MR task scheduling.

To solve the aforementioned problems, we proposed HybridMR, a new MR implementation for hybrid computing environment. Similar to the design of Hadoop, HybridMR is also decomposed into two layers, namely, data storage layer and MR task scheduling and execution layer. First, a hybrid storage system called HybridDFS composed of cluster nodes and volunteer nodes is implemented; then, MR task scheduling is implemented. In order to solve the volatility of volunteer nodes, we designed and implemented a node fault-tolerance mechanism based on the 'heartbeat' and time-out method. Furthermore, an optimized scheduler taking into account performance differences between cluster nodes and volunteer desktop nodes is also implemented.

The rest of the paper is organized as follows. Section 2 surveys research background and related work including MR model and MR on non-dedicated computing resources. Section 3 introduces the system architecture of HybridMR. Section 4 presents the performance evaluation of the prototype system and also the analysis of experimental results. The final section offers concluding remarks.

2. BACKGROUND AND RELATED WORK

2.1. MapReduce

MapReduce model borrows some ideas from functional programming. MR applications are based on a master–slave model. A MR system includes two basic computing units, map and reduce. The MR programming model allows the user to define a map function and a reduce function to realize large-scale data processing and analyzing. In the first step, input data are divided into chunks and distributed in a Distributed File System (DFS), such as Hadoop DFS (HDFS) and Google File System (GFS) [10]. In the second step, Mapper nodes apply the map function on each file chunk. Then, the partition phase achieves splitting the keys space on a mapper node, so that each reducer node gets a part of the key space. This is typically performed by applying a hash function to the keys although programmers can define their own partition function. The new data produced are called the intermediate results. In short, the map function processes a <key, value> pair, and returns a list of intermediate <key, value> pairs:

$$map(k1, v1) \rightarrow list(k2, v2). \quad (1)$$

DOI: 10.1002/cpe

During the shuffle phase, intermediate results are sent to their corresponding reducer. In the reduce phase, reducer nodes apply the reduce function to merge all intermediate values having the same intermediate key:

$$reduce(k2, list(v2)) \rightarrow list(k3, v3). \tag{2}$$

At the end, all the results can be assembled and sent back to the master node, and this is the combine phase.

Currently, many studies have focused on optimizing the performance of MR. Because the common first-in-first-out (FIFO) scheduler in Hadoop MR implementation has some drawbacks and it only considers the homogeneous cluster environments, there are some improved schedulers with higher performance proposed such as fair scheduler, capability scheduler [11], longest approximate time to end (LATE) scheduler [12] deadline Scheduler [13], and constraint-based scheduler [14].

Zaharia *et al.* [12] observed that Hadoop's homogeneity assumptions lead to large number of backup tasks performed and also lead to incorrect and often excessive speculative execution in heterogeneous environments, and can even degrade performance below that obtained with speculation disabled. In some experiments, as many as 80% of tasks were speculatively executed. Therefore, LATE scheduler is designed for heterogeneous Hadoop clusters, which consider the heterogeneity of resources. It starts the backup task for the task that has the longest approximate time to end to improve MR performance in heterogeneous environments.

Xie *et al.* [15] proposed a solution to improve MR performance through data placement in heterogeneous Hadoop clusters. The proposed strategy is placing data across nodes in a way that each node has a balanced data processing load. Xie *et al.* [15] also presented measuring heterogeneity through 'computing ratio' to measure each node's processing speed in a heterogeneous cluster. However, the input and output (I/O) capacity of nodes has not been considered in heterogeneity measurement.

Now, the MR programming model has been successfully applied in many fields such as web data mining, large scale documents analytics, query processing, bioinformatics, financial prediction, social network analytics, recommendation algorithm [16], clustering algorithm [17], privacy-preserving algorithm [18], and so on.

2.2. MapReduce on non-dedicated computing resources

Besides the original MR implementation by Google [4], several other MR implementations have been realized within other systems or environments. Some focused on providing more efficient implementations of MR components such as the scheduler [12] and the I/O system, while others focused on adapting the MR model to specific computing environments like shared-memory systems, graphics processors [19], multi-core systems [20], volunteer computing environments, and desktop grids [5].

BitDew-MapReduce proposed by Tang *et al.* [5] is specifically designed to support MR applications in desktop grids and exploits the BitDew middleware [21], which is a programmable environment for automatic and transparent data management on desktop grid, grid, and cloud. BitDew relies on a specific set of metadata to drive key data management operations, namely, life cycle, distribution, placement, replication, and fault tolerance with a high level of abstraction. Lu *et al.* [22] compared the BitDew-MapReduce implementation with Hadoop, which proved that BitDew-MapReduce outperforms Hadoop in desktop grid environment with node crash. Concerning the results checking, BitDew-MapReduce employs a distributed results checking mechanism that is similar to Berkeley open infrastructure for network computing (BOINC) [23].

Marozzo *et al.* [7] proposed P2P-MapReduce that exploits a peer-to-peer model to manage node churn, master failures, and job recovery in a decentralized but effective way, so as to provide a more reliable MR middleware that can be effectively exploited in dynamic cloud infrastructures.

Another similar work is VMR [9], a volunteer computing system able to run MR applications on top of volunteer resources, spread throughout the Internet. VMR leverages users' bandwidth through the use of inter-client communication and uses a lightweight task validation mechanism. GiGi-MR [8] is another framework that allows nonexpert users to run CPU-intensive jobs on top of volunteer resources over the Internet. Bag-of-tasks are executed in parallel as a set of MR applications.

DOI: 10.1002/cpe

Another system that shares some of the key ideas with HybridMR is MOON [6]. It is a system designed to support MR jobs on opportunistic environments. It extends Hadoop with adaptive task and data scheduling algorithms to offer reliable MR services on a hybrid resource architecture.

The problems and challenges of MR on non-dedicated resources are mainly caused by resource volatile. There are also some work focusing on using node availability prediction method to enable Hadoop running on unreliable desktop grid or non-dedicated computing resources [24, 25].

Concerning about the hybrid computing environment, Antoniu *et al.* [26] first proposed overcoming the limitations of current MR frameworks (such as Hadoop) and enable ultra-scalable MR-based data processing on various physical platforms such as clouds, desktop grids, or on hybrid infrastructures built by combining these two types of infrastructures. It is the initiative of hybrid MR that combines BlobSeer-based [27] cloud and BitDew-based desktop solution, but it did not present complete prototype implementation. Dos Anjos *et al.* [28] implemented a toolkit called BIGhybrid for simulating MR in this kind of hybrid infrastructure. In [29], information dispersal algorithm is adopted to ensure privacy for MR in a hybrid computing environment that is consist of untrusted infrastructures (such as public clouds and desktop grid) and trusted infrastructures (such as private cloud).

3. SYSTEM ARCHITECTURE

In this section, we describe the architecture of HybridMR. First, we present an overview of the system; then, we focus on the implementation of the main components of HybridMR, and we highlight the main scheduling algorithms.

3.1. General overview

HybridMR is composed of reliable cluster nodes and volatile desktop PCs, which are simple but effective. MR applications can be run in this hybrid environment to analyze and process large amounts of datasets. The architecture of HybridMR is shown in Figure 1.

In this figure, the system is designed with a hierarchical architecture. The top layer is the user layer, the middle layer is the service layer, and the bottom layer is the resource layer. Four different service components are implemented in service layer, namely, data storage service, metadata service, data scheduler service, and MR task scheduler service. Resource layer contains two types of resource: the first is reliable cluster nodes (cluster workers) and the second is large number of unreliable volunteer nodes (desktop workers), which join the system in a voluntary way. These two types of resources are both computing and storage resources.

Similar to existing MR systems, data storage layer and MR task-scheduling layer are also separated in our proposed model. The proposed model relies on a hybrid DFS, called HybridDFS, which can also be run independently as a sub-component. HybridDFS has similar characteristics with HDFS and GFS that data are stored in block. The difference is that HybridDFS defines two different types of data storage nodes, the reliable cluster nodes and unreliable volunteer nodes. To sum up, in our proposed model, we implemented

- *ClientNode*, provides interface to access data and submit jobs.
- *NameNode*, provides metadata services.
- *DataNode*, provides data storage services.
- *WorkerNode*, provides MR task computing services.
- *TrackerNode*, provides MR task monitoring services.

Among them, DataNode and WorkerNode can be deployed in cluster nodes or volunteer nodes, while NameNode and TrackerNode can only be configured in server. The main working principle of the system is shown as follows:

- Step 1: ClientNode uploads input data that will be analyzed and processed to HybridDFS.
- Step 2: ClientNode submits task, specifying the data stored in HybridDFS that will be processed.

DOI: 10.1002/cpe

- Step 3: Scheduled by data scheduler and MR scheduler, the map tasks and reduce tasks are allocated to cluster nodes and volunteer nodes. Meanwhile, MR scheduler controls the transmission of intermediate data.
- Step 4: Cluster nodes and volunteer nodes regularly send 'heartbeat' signals to MR scheduler to report task status.
- Step 5: Once all of the tasks have completed, ClientNode can download final results from HybridDFS.

3.2. Design overview of hybrid distributed file system

In this system, the data storage layer is a HybridDFS, which is composed of cluster nodes and volunteer nodes. HybridDFS is a configurable, scalable, and reliable hybrid DFS, which is designed to support MR application in a hybrid computing environment. In HybridDFS, each node contributes a certain space to store files. As we can see in Figure 2, large file is first separated into chunks; then, all chunks are stored in different locations. As volunteer nodes are volatile, the chunks stored in volunteer nodes may become unavailable. Therefore, replication approach is utilized to achieve fault tolerance.

Similar to the design of HDFS, the metadata management system utilizes a centralized method in HybridDFS. The name node manages a server cache, and each data node manages a local cache. HybridDFS maintains a simple file system namespace to support file directory management, stores physical storage location of each copy, and stores the mapping relation between logical file name and physical storage location.

HybridDFS is designed to support large files. A file is split into one or more blocks, and these blocks are stored in a set of data nodes. All blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file, and a typical block size is 64 MB. Users or applications can define the replication factor *Rs*:*Rv* at file creation time in HybridDFS, where *Rs* means the number of replicas

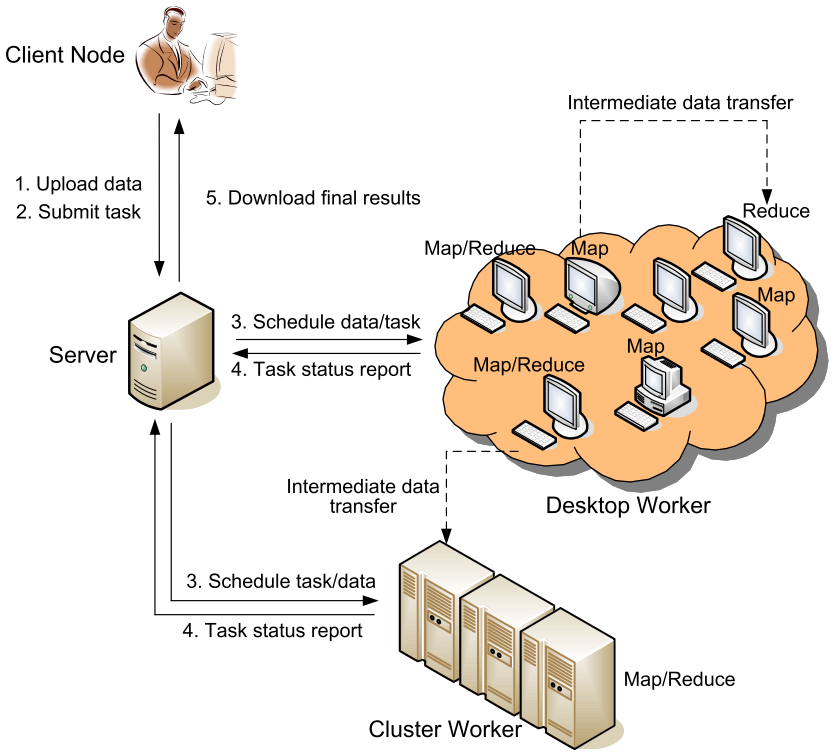


Figure 1. Architecture of hybrid MapReduce computing system.

DOI: 10.1002/cpe

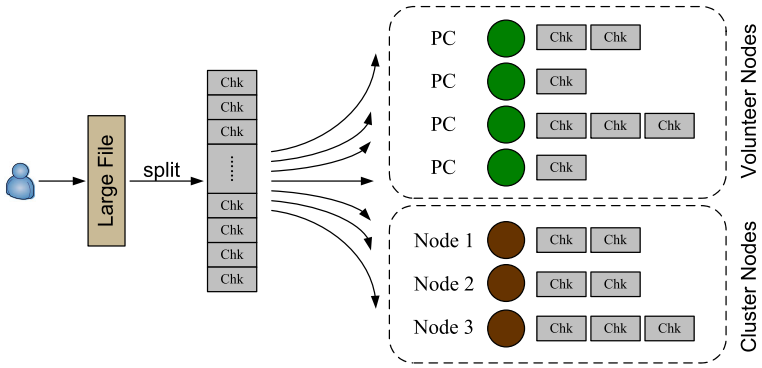


Figure 2. The principle of file separation and storage for large files in HybridDFS.

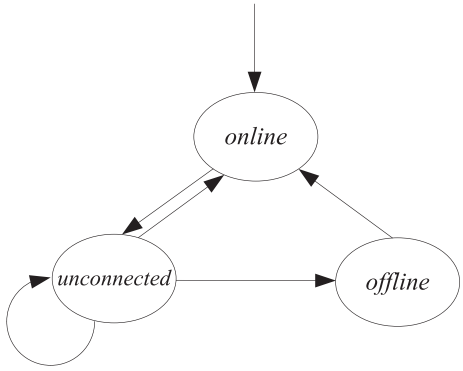


Figure 3. Node status migration chart.

of the file stored in cluster nodes and *Rv* means the number of replicas of the file stored in volunteer nodes. For example, 1:2 means storing one copy in cluster nodes and two copies in volunteer nodes at the same time.

Unlike previous systems, HybridDFS does not differentiate transient failure from permanent failure particularly [30]. We define three node statuses: *online*, *off-line*, and *unconnected*, and the status migration chart is shown in Figure 3. Different with other systems that usually consider the status *dead*, there is a special *unconnected* status. The failure detection is achieved by the method of periodically synchronization ('heartbeat'). The name node makes all decisions regarding replication of blocks. It periodically receives a 'heartbeat' report from each of the data nodes. Receipt of a 'heartbeat' implies that the data node is functioning properly. A report contains a list of all blocks stored in a data node. In order to tolerate node failures, especially the volunteer node failures, HybridDFS uses a *timeout* method to detect node failures. We define two thresholds in this model: synchronization interval time (SIT) and failure timeout time (FTT). If the failure timeout period has expired, a node failure is detected (that becomes *offline*). As the response of 'heartbeat' report, the replicas of file blocks are distributed to different volunteer PCs or cluster nodes.

Volatile nodes declare their availability to the system through periodical synchronization (an interval of SIT) with the server. During each synchronization, the value of variable $alivetime$ is updated to the current time. If the difference between the value of variable $alivetime$ and the current time exceeds FTT, this node becomes *off-line*. The detailed migration of three situations are demonstrated in Figure 4. The white circle stands for periodically node synchronization or node joining to the system, and there is also an updating of the variable $alivetime$ associated with each white circle. The blue circle stands for *unconnected*, while the red circle stands for *off-line* that means a node failure

DOI: 10.1002/cpe

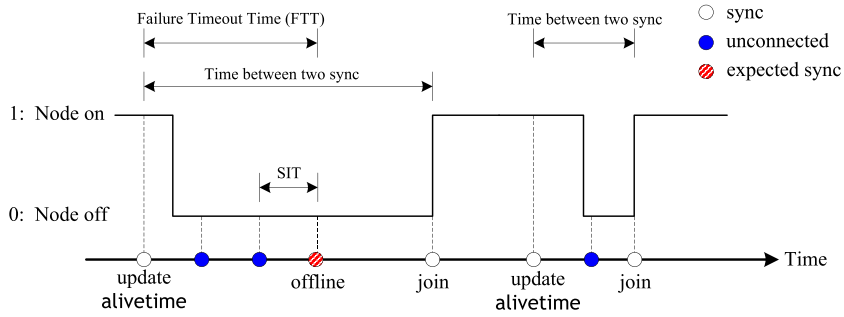


Figure 4. Node synchronization and timeout-based node failure detection method. The detailed migration of three situations: (1) node migration from *online* to *off-line*, (2) node migration from *off-line* to *online*, and (3) re-join (recover) in a short time from *unconnected* to *online*.

is detected. Both the blue circle and red circle indicate that a node synchronization is expected, because the node has already lost the communication with the server.

MapReduce applications demand advanced requirements for HybridDFS. HybridDFS acts as the data storage layer to support running MR tasks. Therefore, HybridDFS encapsulates methods and interfaces, which allow MR applications get to know how the data is separated, and the physical location of blocks can be queried, and the tasks are scheduled to storage nodes by MR scheduler.

3.3. MapReduce algorithm and implementation

The client submits the job that specifies the data to be processed that have already been stored in HybridDFS. By calling HybridDFS application programming interface, data blocking method and the physical storage location of each data block are obtained. According to the file replication attributes *Rs*:*Rv*, one copy of each data block is chosen to run a map task. The large data to be analyzed and processed are denoted by *Data*, which is divided into n blocks, and each block is denoted by d_i. The intermediate results for the selected block when the map task is completed are denoted by IR^i. At the shuffle stage, intermediate results are divided into r groups, and each group is written to HybridDFS. Then, r nodes are selected to run reduce tasks, and each node reads the corresponding intermediate results IR_j from HybridDFS. When all reduce tasks are completed, the final results are denoted by $Output_j$. Throughout all stages in a MR application, n mappers (depending on the number of blocks) and r reducers (defined by the client node when submitting the task) are launched. MR process can be simply described by the following equations:

$$Data = \bigcup_{i=1:n} d_i, \tag{3}$$

$$Map(Data) = \bigcup_{i=1:n} Map(d_i), \tag{4}$$

$$Map(d_i) = \bigcup_{j=1:r} IR_j^i, \tag{5}$$

$$Reduce\left(\bigcup_{i=1:n} IR_j^i\right) = Output_j. \tag{6}$$

In designing the runtime of HybridMR, the general fast/slow nodes detection and fast/slow tasks detection approaches are not fit for this hybrid heterogenous environment, because CPU speed of cluster nodes are always faster than desktop PCs. In existing MR computing models for desktop

DOI: 10.1002/cpe

grid environment such as BitDew-MapReduce [5], the FIFO scheduling policy is usually employed when processing 'heartbeat' report that the data chunks are assigned in the order that 'heartbeat' arrived, without other biases or preferences. In HybridMR implementation, we developed a new node priority-based fair scheduling (NPBFS) algorithm. In the hybrid heterogenous environment, hardware configurations of worker nodes or data nodes are diverse, which proposes an urgent need of a fair algorithm that the node with stronger computing capability should process more jobs.

Therefore, using NPBFS algorithm, the objective is to achieve two kinds of balance in HybridMR: data placement balance (adaptively balances the amount of data stored in each node considering storage capability of each node) and job assignment balance (adaptively balances the task queue length in each node considering computing capability of each node). In HybridMR implementation, job priority is not considered, instead we focus on node priority and developed a node rank method considering the hardware configurations.

We quantify CPU speed, memory size, network bandwidth, and disk I/O speed then calculate the R_{weight} for each data node and worker node, according to the equations as follows:

$$R_{capacity} = \alpha * R_{cpu} * R_{core} + \beta R_{mem} + \delta(R_{network} + R_{disk}), \tag{7}$$

$$R_{storageload} = \sum_{i} (BlockNum[i] \cdot datasize), \tag{8}$$

$$R_{workload} = \sum_{i} (TaskNum[i] \cdot datasize), \tag{9}$$

$$R_{weight} = \frac{R_{workload} + R_{storageload}}{R_{capacity}}, \tag{10}$$

where α, β, and δ are three weight coefficients used to quantify node capacity, $R_{storageload}$ denotes the total size of chunk stored in a data node, and $R_{workload}$ denotes the total size of data to be processed by map tasks and reduce tasks in a worker node, so the value of $R_{workload}$ reflects approximatively the length of task queue. Both cluster nodes and desktop PCs are usually configured as data node and worker node simultaneously; therefore, we use R_{weight} to measure the degree of balance between capacity and load in heterogeneous environment. When a node sends the 'heartbeat' report, the updated R_{weight} value is capsulated in the report. The server receives and stores all R_{weight} values, and all nodes are then sorted by their R_{weight} value. A smaller value of R_{weight} means a higher node priority, and therefore more jobs should be assigned to it, or more file chunks should be placed on it. In this algorithm, R_{cpu} is measured in gigahertz, and R_{mem} is measured in gigabytes, while $R_{network}$ is measured in 100 Mbps, and R_{disk} is measured in 100 MB/s. Both $BlockNum[i] \cdot datasize$ and $TaskNum[i] \cdot datasize$ are measured in gigabytes.

We define a threshold Th_{weight} to distinguish overloaded nodes as follows:

$$Th_{weight} = \xi \left[\max(R_{weight}[j]) - \min(R_{weight}[j])\right] + \min(R_{weight}[j]) \tag{11}$$

where ξ is an adjustment factor. When a node sends the 'heartbeat', if $R_{weight} > Th_{weight}$, the server must stop placing new chunks or allocating new map/reduce tasks to this overloaded node; otherwise, it means that this is not an overloaded node that can accept more jobs.

The detailed algorithms for data block distribution, map task scheduling, and reduce task scheduling are described in Algorithm 1, 2, and 3, respectively.

4. PERFORMANCE EVALUATION

4.1. Platform description

The prototype system of HybridMR is implemented by Java. In order to evaluate the performance, we performed our experiments in the campus local area network environment, and Hadoop-0.21.0 is used for comparison. Both HybridMR and Hadoop ran on Ubuntu Linux system 13.10 (Canonical Ltd., London, UK). In order to evaluate NPBFS algorithm, the parameters are set to

DOI: 10.1002/cpe

Algorithm 1 DATA BLOCK DISTRIBUTION ALGORITHM

Require: **Let** $D = \{d_i | i = 1, 2, 3, \cdots, n\}$ be the file which is separated into n blocks
Require: **Let** R_s:R_v be the replication factor
Require: **Let** $.sno$ be the number of replicas that are stored in cluster nodes
Require: **Let** $.vno$ be the number of replicas that are stored in volunteer nodes
Require: **Let** Ψ_{data_ta} be the set of data blocks to be assigned
Require: **Let** Θ be the set of data blocks assigned to a DataNode

1. {on the Server node, system initialization}
2. **for all** DataNode in HybridDFS **do**
3. $\Theta \leftarrow \Phi$
4. **end for**
5.
6. {on the Server node, when the client writes a file D to HybridDFS}
7. $\Psi_{data\_ta} \leftarrow \{d_i | i = 1, 2, 3, \cdots, n\}$
8. **for all** data block $d_i \in \Psi_{data\_ta}$ **do**
9. $d_i.sno \leftarrow 0$
10. $d_i.vno \leftarrow 0$
11. **end for**
12.
13. {on the DataNode}
14. periodically calculate R_{weight}, and send heartbeat report
15.
16. {on the Server node, when it receives a heartbeat report from a DataNode}
17. update the threshold Th_{weight}
18. **if** NPBFS is enabled **and** $R_{weight} > Th_{weight}$ **then**
19. stop distributing block to this DataNode
20. **else**
21. **if** the DataNode is a cluster node **then**
22. find a data block x ($x \in \Psi_{data\_ta}$ and $x.sno < R_s$)
23. assign block x to this DataNode
24. $x.sno \leftarrow x.sno + 1$
25. $\Theta \leftarrow \Theta \cup x$
26. **if** $x.sno == R_s$ **and** $x.vno == R_v$ **then**
27. distributing block x finished
28. $\Psi_{data\_ta} \leftarrow \Psi_{data\_ta} \backslash x$
29. **if** $\Psi_{data\_ta} == \Phi$ **then**
30. distributing data D finished
31. **end if**
32. **end if**
33. **else**
34. find a data block y ($y \in \Psi_{data\_ta}$ and $y.vno < R_v$)
35. assign block y to this DataNode
36. $y.vno = y.vno + 1$
37. $\Theta \leftarrow \Theta \cup y$
38. **if** $y.sno == R_s$ **and** $y.vno == R_v$ **then**
39. distributing block y finished
40. $\Psi_{data\_ta} \leftarrow \Psi_{data\_ta} \backslash y$
41. **if** $\Psi_{data\_ta} == \Phi$ **then**
42. distributing data D finished
43. **end if**
44. **end if**
45. **end if**
46. **end if**

empirical values. Three weight coefficients α, β, and δ are set to 0.4, 0.2, and 0.4, respectively, and the value of adjustment factor ξ is 0.6. Our experimental hardware platforms are described as follows:

(1) Both the NameNode and TrackerNode are configured with Intel (Intel, Santa Clara, CA, USA) Xeon E5 2603 quad-core 1.8 GHz CPU, 4 GB memory, and 1 Gbps ethernet. The average disk read/write speed is around 96 MB/s.

DOI: 10.1002/cpe

Algorithm 2 SCHEDULING ALGORITHM IN MAP STAGE

Require: **Let** $D = \{d_i | i = 1, 2, 3, \cdots, n\}$ be the file which is separated and stored in HybridDFS
Require: **Let** Θ be the set of data blocks assigned and stored in a DataNode/WorkerNode
Require: **Let** Ψ_{map_ta} be the set of input data to be assigned
Require: **Let** Ψ_{reduce_ta} be the set of intermediate results to be assigned
Require: **Let** Ω_{map_tasks} be the set of Map input data assigned to a WorkerNode
Require: **Let** Ω_{reduce_tasks} be the set of intermediate results assigned to a WorkerNode

1. {on the Server node, system initialization}
2. **for all** WorkerNode in HybridMR **do**
3. $\Omega_{map\_tasks} \leftarrow \Phi$
4. $\Omega_{reduce\_tasks} \leftarrow \Phi$
5. **end for**
6.
7. {on the Server node, when the client submits a job that specifies data D to be processed}
8. $\Psi_{map\_ta} \leftarrow \{d_i | i = 1, 2, 3, \cdots, n\}$
9.
10. {on the WorkerNode}
11. periodically calculate R_{weight} through Θ, Ω_{map_tasks} and Ω_{reduce_tasks}, and send heartbeat report
12.
13. {on the Server node, when it receives a heartbeat report from a WorkerNode}
14. update the threshold Th_{weight}
15. **if** NPBFS is enabled **and** $R_{weight} > Th_{weight}$ **then**
16. stop assigning Map tasks to this WorkerNode
17. **else**
18. {lookup data blocks in Θ of this WorkerNode}
19. **if** there is a data block b ($b \in \Theta$ **and** $b \in \Psi_{map\_ta}$) **then**
20. notify this WorkerNode to start Mapper to process data block b
21. $\Psi_{map\_ta} \leftarrow \Psi_{map\_ta} \backslash b$
22. **end if**
23. **if** $\Psi_{map\_ta} == \Phi$ **then**
24. all Map tasks associated with data D are assigned
25. **end if**
26. **end if**
27.
28. {on the WorkerNode, when it receives the notification of starting Mapper to process data block b}
29. $\Omega_{map\_tasks} \leftarrow \Omega_{map\_tasks} \cup b$
30. $\{IR_j | j = 1, 2, 3, \cdots, r\} \leftarrow Map(b)$
31. write IR_j to HybridDFS
32. $\Psi_{reduce\_ta} \leftarrow \Psi_{reduce\_ta} \cup \{IR_j | j = 1, 2, 3, \cdots, r\}$

(2) We used 24 cluster nodes, and each node is configured with AMD (AMD, Sunnyvale, CA, USA) Opteron 8378 quad-core 2.4 GHz CPU, 8 GB memory, and 1 Gbps ethernet. The average disk read/write speed reaches 88 MB/s.
(3) In the students' laboratory, we used 72 desktop PCs, configured with Intel (Intel, Santa Clara, CA, USA) core 2 duo E6300 1.86 GHz CPU, 1 GB memory, and 100 Mbps ethernet for each. The average disk read/write speed is around 69 MB/s.

4.2. Throughput of HybridDFS input and output

We have implemented a set of micro-benchmarks and have measured the achieved throughput as more and more concurrent clients access HybridDFS. Because MR applications need the 'write-once-read-many' model, we evaluated the I/O performance when a single client writes data and concurrent clients read data. We also compared HybridDFS with HDFS.

4.2.1. Scenario 1: single writer, single file. We first measure the performance of HybridDFS when a single client writes a file whose size gradually increases. The size of data chunks in HybridDFS is 64 MB. This test consists in sequentially writing a unique file of $N * 64$ MB (N goes from 1 to 192). Block allocation is also based on the node priority-based fair scheduling policy that is explained

DOI: 10.1002/cpe

Algorithm 3 SCHEDULING ALGORITHM IN REDUCE STAGE

Require: **Let** $D = \{d_i | i = 1, 2, 3, \cdots, n\}$ be the file which is separated and stored in HybridDFS
Require: **Let** Θ be the set of data blocks assigned and stored in a DataNode/WorkerNode
Require: **Let** Ψ_{reduce_ta} be the set of intermediate results to be assigned
Require: **Let** Ω_{map_tasks} be the set of Map input data assigned to a WorkerNode
Require: **Let** Ω_{reduce_tasks} be the set of intermediate results assigned to a WorkerNode

1. {on the WorkerNode which is selected to execute Reduce tasks}
2. periodically calculate R_{weight} through Θ, Ω_{map_tasks} and Ω_{reduce_tasks}, and send heartbeat report
3.
4. {on the Server node, when it receives a heartbeat report from a Reducer WorkerNode}
5. update the threshold Th_{weight}
6. **if** NPBFS is enabled **and** $R_{weight} > Th_{weight}$ **then**
7. stop assigning Reduce tasks to this WorkerNode
8. **else**
9. **if** the WorkerNode has not a key **then**
10. assign a key to this WorkerNode
11. $IR^{key} \leftarrow \Phi$
12. **end if**
13. notify this WorkerNode to start Reduce task to process intermediate results associated with key
14. **if** $\Psi_{reduce\_ta} == \Phi$ **then**
15. all Reduce tasks associated with data D are assigned
16. **end if**
17. **end if**
18.
19. {on the WorkerNode which is selected to execute Reduce tasks}
20. download intermediate results IR^{key} directly from DataNodes after accessing metadata server
21. $\Omega_{reduce\_tasks} \leftarrow \Omega_{reduce\_tasks} \cup IR^{key}$
22. $\Psi_{reduce\_ta} \leftarrow \Psi_{reduce\_ta} \backslash IR^{key}$
23. $\{Output_j | j = 1, 2, 3, \cdots, r\} \leftarrow Reduce(IR^{key})$
24. write $Output_j$ to HybridDFS

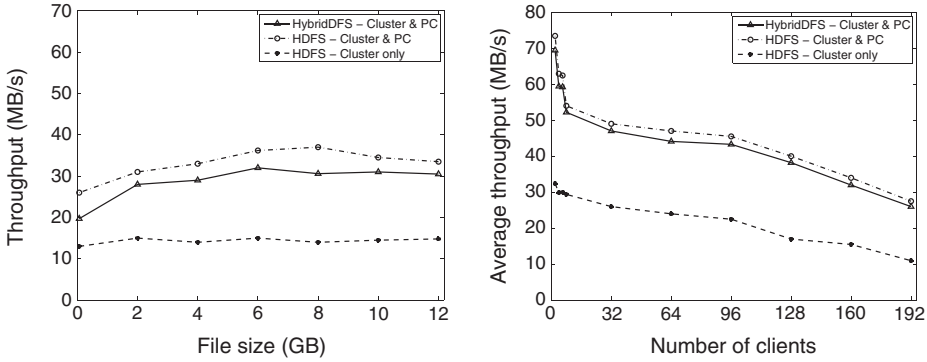
before, in order to achieve placing data across data nodes in balance. We measure the time spend for file separation and file distribution and then calculate the write throughput in three different conditions:

- HybridDFS – 24 cluster nodes and 72 desktop PCs.
- HDFS – 24 cluster nodes and 72 desktop PCs.
- HDFS – 24 cluster nodes only.

The results can be seen in Figure 5(a). The value of SIT and FTT are set to 10s and 30s, respectively. The file replication attribute setting is *Rs*:*Rv*=1:2, which means that storing one copy in cluster nodes and two copies in desktop nodes. Therefore, the total number of blocks of a large file stored in HybridDFS is *N**3. As the file size increases, the change of throughput is very tiny. Obviously, we obtain the worst results when only 24 cluster nodes are used, and HDFS achieves higher throughput than HybridDFS when 24 cluster nodes and 72 desktop PCs are used. The main reason why HybridDFS is inferior to HDFS is that NPBFS realizes storage balance among all storage nodes. According to Equation (10) and (11), when the client writes a file, as the increase of the amount of data stored in one node increases, the value of $R_{storageload}$ and R_{weight} change at the same time. The node may become overloaded, and new chunks will not be distributed to it. Therefore, NPBFS delays the write client.

4.2.2. Scenario 2: concurrent readers, shared file. In this scenario, *N* clients read parts from the file concurrently; each client reads different 64 MB chunks. This pattern where multiple readers request data is very common in the Map stage of Hadoop MapReduce application, where the Mappers read the input file in order to parse <key, value> pairs. When a single client finished writing a file of 192*64 MB to HybridDFS, for each given number *N* of clients varying from 1 to 192, we executed the experiments and calculated the average throughput. The total size of chunks read by *N* clients is exactly 192*64 MB. Figure 5(b) shows the results of average throughput of concurrent read clients.

DOI: 10.1002/cpe



(a) Throughput of single write client. (b) Average throughput of multi-read clients.

Figure 5. Throughput of HybridDFS input and output (I/O), (a) throughput of single write client and (b) average throughput of multi-read clients. HDFS, Hadoop distributed file system; PC, personal computer.

When the number of concurrent clients is more than 64, less than 3 chunks are allocated to each client in average. As the increase of concurrent read clients, the metadata query load and data traffic increases, which causes a decrease of average throughput. The same as Scenario 1, HDFS also outperforms HybridDFS when 24 cluster nodes and 72 desktop PCs are used, but there is only little difference between average throughput of HybridDFS and HDFS. HybridDFS reaches a relatively high throughput.

4.3. MapReduce job completion time

In order to evaluate how well HybridMR performs in real MR applications, we select two standard MR applications *word count* (reads text files and counts how often words occur) and *distributed grep* (extracts matching strings from text files and counts how many times they occurred). For these two applications, the chunk size of input text files is still 64 MB.

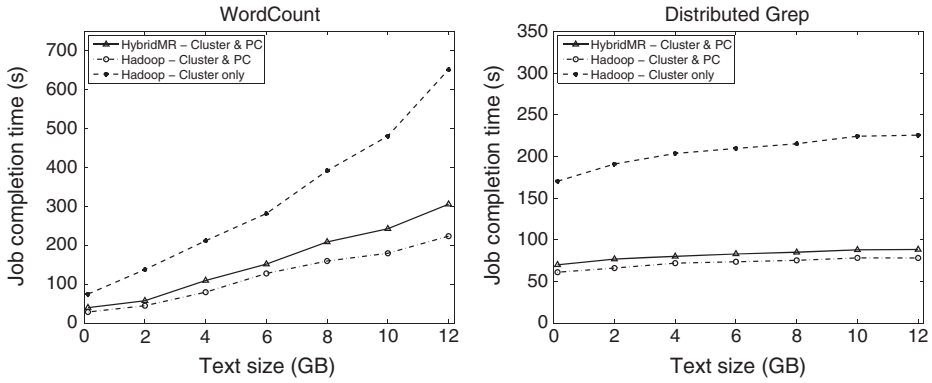
We evaluated MR job completion time as the size of input text file changes. The same as read/write throughput evaluation, in order to compare HybridMR with Hadoop, we also measured three conditions just like before. We have not configured any scheduling strategy in Hadoop, and jobs are scheduled in default FIFO order.

The results for word count and distributed grep are shown in Figures 6(a) and 6(b), respectively. The maximal size of text file is 12 GB in our experiments. Distributed grep application has a different MR pattern compared with word count application. For word count application, the reduce stage is complex and takes more time than the map stage. For distributed grep application, the reduce stage is very simple, and it just collects and sums up the intermediate results. As you can see from these two figures, as the increase of input text file size, job completion time also increases. From these two figures, we can see that there is also only little performance difference between HybridMR and Hadoop.

4.4. Scheduler optimization

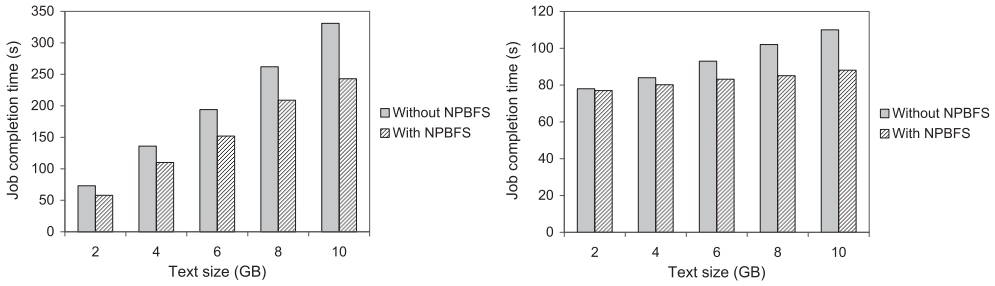
In this scenario, experiments on word count application and distributed grep application have also been performed to testify the efficiency of the NPBFS algorithm. We also evaluated job completion time, while we compared two scheduling policies: (1) using NPBFS scheduler and (2) not using NPBFS scheduler. We measure how many performance improvements are caused by NPBFS scheduler. If the NPBFS scheduler is not used, the server does not consider any information or attributes of nodes, and all nodes are treated equally, which may cause the problem that assigning a lot of tasks to slow desktop PCs. HybridMR is deployed on a hybrid environment composed of 24 cluster nodes and 72 desktop PCs; then, we run word count and distributed grep again and measure the job completion time, varying the input text file size from 2 to 10 GB. The results are shown in Figures 7(a) and 7(b), respectively. These two figures indicate that NPBFS scheduler improves the

DOI: 10.1002/cpe



(a) WordCount application (b) Distributed Grep application

Figure 6. Job completion time for word count and distributed grep: (a) word count application and (b) distributed grep application. HybridMR, hybrid MapReduce; PC, personal computer.



(a) WordCount application (b) Distributed Grep application

Figure 7. Performance improvements when the node priority-based fair scheduling policy is used: (a) word count application and (b) distributed grep application. NPBFS, new node priority-based fair scheduling.

whole system and makes it more balanced, and decreases the overall job response time. When the text file size is 10 GB, the performance improvement is 26.6% for word count, while it is 19.9% for distributed grep.

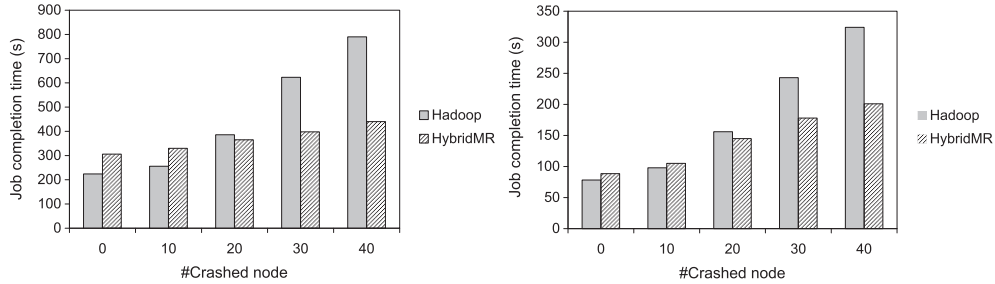
There are two popular pluggable schedulers in Hadoop. Unlike the default Hadoop FIFO scheduler, fair scheduler provides a way to share large clusters for multi-jobs while maximizing the throughput and the utilization of the cluster, while capacity scheduler allows for multiple tenants to securely share a large cluster such that their applications are allocated resources in a timely manner under constraints of allocated capacities. In HybridMR, the proposed NPBFS scheduler considers only the node priority, and we have not implemented the fair scheduling and capacity scheduling strategy for multi-jobs. Jobs are performed according to the arrival time, and in the FIFO order.

4.5. Fault tolerance

In this scenario, we compare HybridMR with Hadoop in terms of fault-tolerance performance, in order to justify the robustness of HybridMR. We emulate node crashes through generating failures by randomly selecting desktop PCs and killing the MR process, during the MR tasks execution period. Failures are independents and occur sequentially. During the experiment, both Hadoop and HybridMR are deployed on a hybrid environment composed of 24 cluster nodes and 72 desktop PCs. We run the word count and distributed grep, which represents two different realistic situations, and the input text file size is 12 GB. The results are shown in Figures 8(a) and 8(b), respectively.

When the number of failures injected are varied from 10 to 40, we measure the job completion time, which is then compared with the normal situation that without any failures. The interval between two failure injections is 60 s. We observe that HybridMR outperforms Hadoop in terms of fault-tolerance performance. Compared with the normal situation, in the worst situation that 40

DOI: 10.1002/cpe



(a) WordCount application (b) Distributed Grep application

Figure 8. Fault-tolerance performance comparison between HybridMR and Hadoop: (a) word count application and (b) distributed grep application.

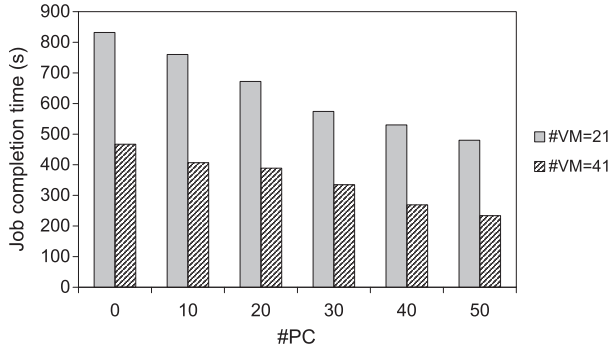


Figure 9. Job completion time when different number of desktop PCs are used as workers.

nodes are crashed, for word count application, the job completion time increases by around 252.7% for Hadoop and only 43.8% for HybridMR; for distributed grep application, it increases by around 313.8% for Hadoop and only 127.1% for HybridMR. The improvement of HybridMR over Hadoop in terms of fault-tolerance performance is quite clear when the number of failure injected is beyond 30. This reveals the robustness of HybridMR, which can accept a large number of faults with a reasonable performance overhead.

4.6. Cost-saving evaluation

In this scenario, we demonstrate how does HybridMR save cost for users through using large number of idle desktop PCs. Similar to Amazon EC2, we build a private on-demand infrastructure-as-a-service cloud using OpenStack in the French grid'5000[‡] experimental platform, which is worldwide recognized as innovative and useful grid/cloud test platform [31]. We show that HybridMR can cause cost-saving in hybrid heterogeneous environments composed of cloud infrastructure and desktop PCs, and present a simple analysis of cost-saving.

In the first phase, we demand 21 virtual machines (one virtual machine (VM) is deployed as the server, and other 20 VMs are deployed as workers) in the cloud. We run the word count application, and the input text file size is 12 GB. We measure the job completion time in the cloud. Then, the number of desktop PCs is gradually increased from 10 to 50, and we measure the job completion time in the hybrid environment. In the second phase, we do the same test varying the number of desktop PCs, but we demand 41 virtual machines (one VM is deployed as the server, and other 40 VMs are deployed as workers). The result of job completion time is shown in Figure 9.

[‡]http://www.grid5000.fr.

DOI: 10.1002/cpe

Table I. Cost-saving rates when different number of desktop PCs are utilized as workers.

number of PCs	10	20	30	40	50
VM = 21	8.7%	19.2%	31.0%	36.2%	42.3%
VM = 41	12.8%	16.7%	28.3%	42.4%	49.9%

PC, personal computer.

Similar to Amazon EC2, cost spent for word count application can be calculated using the pricing model. To simplify the problem, cost-saving rate can be estimated only using the time as the following equation:

$$Rate_{cost\text{-}saving} = \frac{T_{cloud} - T_{cloud+pc}}{T_{cloud}} \times 100\%, \tag{12}$$

where T_{cloud} means the job completion time when VMs are deployed as workers, while $T_{cloud+pc}$ means the job completion time when both VMs and desktop PCs are deployed as workers.

As can be seen from Figure 9, the job completion time decreases conspicuously, caused by desktop PCs joining. Correspondingly, Table I demonstrates the cost-saving rates in different situations. We observe from this table that when the number of virtual machines are 21 and 41, the cost-saving rate reaches 42.3% and 49.9%, respectively, after 50 desktop PCs are utilized as workers at nearly zero-cost. Utilizing idle desktop PCs as workers, HybridMR can improve job response time rapidly and save cost of cloud services.

5. CONCLUSION

This paper presented a MR parallel model for data-intensive computing in dynamic hybrid computing environments, integrating the idle desktop PC resources in the Internet or Intranet with high reliable and high performance cluster nodes to form a hybrid computing environment. The proposed new MR model consists of HybridDFS layer, a new hybrid DFS, and MR task-scheduling layer. Data replication and replacement mechanism are utilized to guarantee the reliability of storage and computing. Security issues will be considered in the future. Performance test results show that the new model is not only able to achieve a higher throughput and efficiency but also able to achieve the 'green computing' goal. Companies and schools can leverage existing idle desktop PC resources running a MR job for massive data analysis, and the proposed method also reduces the computational cost overhead, which has a great potential.

ACKNOWLEDGEMENTS

This work is supported by the French Agence Nationale de la Recherche through the MapReduce grant under contract ANR-10-SEGI-001-01 and INRIA ARC BitDew. This work is also supported by the '100 Talents Project' of Computer Network Information Center of Chinese Academy of Sciences under grant number 1101002001, the Natural Science Foundation of Hunan Province under grant number 2015JJ3071, and the Scientific Research Fund of Hunan Provincial Education Department under grant number 12C0121.

Some experiments presented in this paper were carried out using the grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER, and several universities as well as other funding bodies (see https://www.grid5000.fr).

REFERENCES

1. Anderson DP. Boinc: A system for public-resource computing and storage. In *GRID*. IEEE: Piscataway, NJ, USA, 2004; 4–10.
2. Cappello F, Djilali S, Fedak G, Hérault T, Magniette F, Néri V, Lodygensky O. Computing on large-scale distributed systems: xtremweb architecture, programming models, security, tests and convergence with grid. *Future Generation Computer Systems* 2005; **21**(3):417–437.
3. Litzkow MJ, Livny M, Mutka MW. Condor – a hunter of idle workstations. *ICDCS*, San Jose, California, USA, 1988; 104–111.

DOI: 10.1002/cpe

4. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. *Communications of the ACM* 2008; **51**(1):107–113.
5. Tang B, Moca M, Chevalier S, He H, Fedak G. Towards MapReduce for desktop grid computing. In *3PGCIC*, Xhafa F, Barolli L, Nishino H, Aleksy M (eds). IEEE Computer Society: Piscataway, NJ, USA, 2010; 193–200.
6. Lin H, Ma X, chun Feng W. Reliable MapReduce computing on opportunistic resources. *Cluster Computing* 2012; **15**(2):145–161.
7. Marozzo F, Talia D, Trunfio P. P2p-MapReduce: parallel data processing in dynamic cloud environments. *Journal of Computer and System Sciences* 2012; **78**(5):1382–1402.
8. Costa F, Silva JN, Veiga L, Ferreira P. Large-scale volunteer computing over the internet. *Journal of Internet Services and Applications* 2012; **3**(3):329–346.
9. Costa F, Veiga L, Ferreira P. Internet-scale support for map-reduce processing. *Journal of Internet Services and Applications* 2013; **4**(1):1–17.
10. Ghemawat S, Gobioff H, Leung ST. The google file system. In *SOSP*. ACM: New York, NY, USA, 2003; 29–43.
11. Zaharia M, Borthakur D, Sen Sarma J, Elmeleegy K, Shenker S, Stoica I. Job scheduling for multi-user MapReduce clusters. *Technical Report UCB/EECS-2009-55*, EECS Department, University of California: Berkeley, 2009.
12. Zaharia M, Konwinski A, Joseph AD, Katz RH, Stoica I. Improving MapReduce performance in heterogeneous environments. In *OSDI*. USENIX Association: Berkeley, CA, USA, 2008; 29–42.
13. Polo J, Carrera D, Becerra Y, Steinder M, Whalley I. Performance-driven task co-scheduling for MapReduce environments. *IEEE/IFIP Network Operations and Management Symposium, NOMS 2010*, Osaka, Japan, 2010; 373–380.
14. Kc K, Anyanwu K. Scheduling Hadoop jobs to meet deadlines. *CloudCom 2010*, Indianapolis, Indiana, USA, 2010; 388–392.
15. Xie J, Yin S, Ruan X, Ding Z, Tian Y, Majors J, Manzanares A, Qin X. Improving MapReduce performance through data placement in heterogeneous Hadoop clusters. In *IPDPS Workshops*. IEEE: Piscataway, NJ, USA, 2010; 1–9.
16. Shang Y, Li Z, Qu W, Xu Y, Song Z, Zhou X. Scalable collaborative filtering recommendation algorithm with MapReduce. *IEEE 12th International Conference on Dependable, Autonomic and Secure Computing, DASC 2014, Dalian, China, August 24-27, 2014*, Dalian, China, 2014; 103–108.
17. Xu Y, Qu W, Li Z, Min G, Li K, Liu Z. Efficient k-means++ approximation with MapReduce. *IEEE Transactions on Parallel and Distributed Systems* 2014; **25**(12):3135–3144.
18. Zhang X, Yang LT, Liu C, Chen J. A scalable two-phase top-down specialization approach for data anonymization using MapReduce on cloud. *IEEE Transactions on Parallel and Distributed Systems* 2014; **25**(2):363–373.
19. He B, Fang W, Luo Q, Govindaraju NK, Wang T. Mars: a MapReduce framework on graphics processors. In *PACT*, Moshovos A, Tarditi D, Olukotun K (eds). ACM: New York, NY, USA, 2008; 260–269.
20. Ranger C, Raghuraman R, Penmetsa A, Bradski GR, Kozyrakis C. Evaluating MapReduce for multi-core and multiprocessor systems. In *HPCA*. IEEE Computer Society: Piscataway, NJ, USA, 2007; 13–24.
21. Fedak G, He H, Cappello F. BitDew: a data management and distribution service with multi-protocol file transfer and metadata abstraction. *Journal of Network and Computer Applications* 2009; **32**(5):961–975.
22. Lu L, Jin H, Shi X, Fedak G. Assessing mapreduce for internet computing: a comparison of Hadoop and BitDew-MapReduce. In *GRID*. IEEE Computer Society: Piscataway, NJ, USA, 2012; 76–84.
23. Moca M, Silaghi GC, Fedak G. Distributed results checking for MapReduce in volunteer computing. In *IPDPS Workshops*. IEEE: Piscataway, NJ, USA, 2011; 1847–1854.
24. Jin H, Yang X, Sun XH, Raicu I. Adapt: availability-aware MapReduce data placement for non-dedicated distributed computing. In *ICDCS*. IEEE: Piscataway, NJ, USA, 2012; 516–525.
25. Lee K, Figueiredo RJO. MapReduce on opportunistic resources leveraging resource availability. *CloudCom*, Taipei, Taiwan, 2012; 435–442.
26. Antoniu G, Costan A, Bigot J, Desprez F, Fedak G, Gault S, Pérez C, Simonet A, Tang B, Blanchet C, Terreux R, Bougé L, Briant F, Cappello F, Keahey K, Nicolae B, Suter F. Scalable data management for Map-Reduce-based data-intensive applications: a view for cloud and hybrid infrastructures. *IJCC* 2013; **2**(2/3):150–170.
27. Nicolae B, Antoniu G, Bougé L, Moise D, Carpen-Amarie A. Blobseer: next-generation data management for large scale infrastructures. *Journal of Parallel and Distributed Computing* 2011; **71**(2):169–184.
28. Dos Anjos JCS, Fedak G, Geyer CFR. Bighybrid – a toolkit for simulating MapReduce in hybrid infrastructures. *2014 International Symposium on Computer Architecture and High Performance Computing Workshop (SBAC-PADW)*, Paris, France, 2014; 132–137.
29. Cheikh AB, Abbes H, Fedak G. Towards privacy for MapReduce on hybrid clouds using information dispersal algorithm. *Data Management in Cloud, Grid and P2P Systems - 7th International Conference, Globe 2014, Munich, Germany, September 2-3, 2014. Proceedings*, Munich, Germany, 2014; 37–48.
30. Tang B, Fedak G. Analysis of data reliability tradeoffs in hybrid distributed storage systems. In *IPDPS Workshops*. IEEE Computer Society: Piscataway, NJ, USA, 2012; 1546–1555.
31. Bolze R, Cappello F, Caron E, Daydé MJ, Desprez F, Jeannot E, Jégou Y, Lanteri S, Leduc J, Melab N, Mornet G, Namyst R, Primet P, Quétier B, Richard O, Talbi EG, Touche I. Grid'5000: a large scale and highly reconfigurable experimental grid testbed. *IJHPCA* 2006; **20**(4):481–494.

DOI: 10.1002/cpe