Future Generation Computer Systems 53 (2015) 43–54

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Locality and loading aware virtual machine mapping techniques for optimizing communications in MapReduce applications

Ching-Hsien Hsu [a,b,*], Kenn D. Slagter [c], Yeh-Ching Chung [c]

[a] Department of Computer Science, Chung Hua University, Hsinchu, 300, Taiwan, ROC

[b] School of Computer and Communication Engineering, Tianjin University of Technology, Tianjin, 300384, PR China

[c] Department of Computer Science, National Tsing Hua University, Hsinchu, 300, Taiwan, ROC

HIGHLIGHTS

- Improving performance of MapReduce programs in heterogeneous environments and hybrid clouds.
- Enhancing data locality through a virtual machine mapping technique.
- Optimizing shuffle performance and reducing communication overheads in distributed systems.
- We propose a loading aware technique to balance workload of reducers at run-time.

ARTICLE INFO

Article history:
Received 9 January 2014
Received in revised form
15 October 2014
Accepted 16 April 2015
Available online 1 June 2015

Keywords:
BigData
Cloud computing
Distributed computing
Heterogeneity
MapReduce
Virtual machines

ABSTRACT

Big data refers to data that is so large that it exceeds the processing capabilities of traditional systems. Big data can be awkward to work and the storage, processing and analysis of big data can be problematic. MapReduce is a recent programming model that can handle big data. MapReduce achieves this by distributing the storage and processing of data amongst a large number of computers (nodes). However, this means the time required to process a MapReduce job is dependent on whichever node is last to complete a task. Heterogeneous environments exacerbate this problem.

In this paper we propose a method to improve MapReduce execution in heterogeneous environments. This is done by dynamically partitioning data before the Map phase and by using virtual machine mapping in the Reduce phase in order to maximize resource utilization. Simulation and experimental results show an improvement in MapReduce performance, including data locality and total completion time with different optimization approaches.

1. Introduction

Big Data is relative term that refers to datasets that have grown to a size that is awkward to work as conventional software tools to capture, manage and process in a tolerable period of time [1]. The source of this data is wide and varied. Typical examples include RFID tags, GPS-enabled smart phones, social media, phone records, web logs, sensor networks, online browsing, eCommerce, and in various scientific research such as astronomy, medicine and weather [2–4]. By mining this data researchers can discover trends such as user behavior. Such knowledge can have an impact on businesses, government, and scientific endeavors.

MapReduce is a programming model to create distributed applications that can process big data using a large number of commodity computers. Originally developed by Google, MapReduce enjoys wide use by both industry and academia [5] via Hadoop [6]. Hadoop is an open source implementation of MapReduce developed by Yahoo and is based on Google's MapReduce [7] and Google File System [8] papers.

The advantages of MapReduce framework is that it allows users to execute analytical tasks over big data without worrying about the myriad of details inherent in distributed programming [5,9]. Both scalable and fault tolerant MapReduce frameworks potentially reduce the time it takes to complete a job by an amount that is proportionally related to the number of nodes available. The efficacy of MapReduce can be undermined however by its implementation. For instance, Hadoop the most popular open source

* Correspondence to: Department of Computer Science and Information Engineering, Chung Hua University, Hsinchu, 300, Taiwan, ROC.
E-mail addresses: chh@chu.edu.tw (C.-H. Hsu), kennslagter@sslab.cs.nthu.edu.tw (K.D. Slagter), ychung@cs.nthu.edu.tw (Y.-C. Chung).

http://dx.doi.org/10.1016/j.future.2015.04.006
0167-739X/

MapReduce framework [9] assumes all the nodes in the network to be homogeneous. Consequently, Hadoop's performance is not optimal in a heterogeneous environment.

In the MapReduce model the time it takes to complete a job depends on when each node completes its workload. Therefore, if the workload is distributed evenly, the slowest node determines the time a job completes. To compensate for this the workload on slower nodes needs to be less than the faster nodes. This can be achieved by dividing the workload proportionally amongst individual nodes based on the processing efficiency of each node.

In this paper we focus on the Hadoop framework. We look in particular how MapReduce handles map input and reduce task assignment in a heterogeneous environment. This is important area of research since there is ample opportunity for MapReduce to be deployed in such environments. For instance, as technology advances, new machines on the network are likely be dissimilar to old ones. Alternatively, MapReduce may be deployed on a hybrid cloud environment, where computing resources tend to be heterogeneous. Therefore, this paper proposes a method to improve execution of MapReduce jobs in a heterogeneous environment.

In summary, this paper presents the following contributions

- A method to improve mapper performance in a heterogeneous environment by repartitioning data at each node.
- A method to improve virtual machine mapping for reducers.
- A method to improve reducer selection on a heterogeneous systems.

The rest of this paper is organized as follows. In Section 2, we present some background on Map Reduce. In Section 3, we present our proposed dynamic data partitioning and virtual machine mapping methods. In Section 4, we evaluate our work, present our experimental results and discuss our findings. In Section 5, we discuss related work. Finally, in Section 6, we present our conclusion and give a brief discussion of future work.

2. Background

The term cloud computing appears to have been coined by Google's CEO, Eric Shmidt at a conference in 2006, and is likely inspired by the use of a cloud to represent the Internet in pictures and diagrams [10]. Amongst the literature [10–13] it appears that there is no standard definition of what cloud computing is. Therefore, this paper uses the NIST definition [14] of cloud computing being "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction".

The introduction of cloud computing presents the concept of utility computing. Utility computing provides consumer-computing resources as a service. Utility computing treats computing as a metered service, like electricity or natural gas [11]. It provides consumers access to computing resources dependent on their needs from providers. This benefits users, as they need not invest capital to build and maintain their own data center. It also gives customers the ability to cope with unexpected demands for resources, and only pay for resources they need during non-peak periods [11].

In this paper, there are two important enabling technologies used by cloud computing, virtualization and MapReduce. Virtualization provides cloud computing's flexible and scalable hardware services [12]. Meanwhile, MapReduce is a programming model to process and access large datasets in a distributed programming environment.

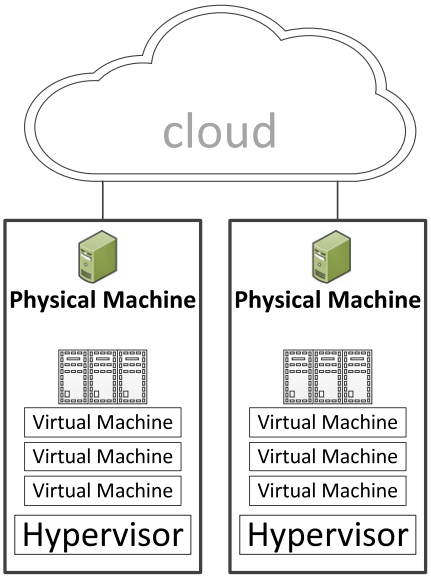

Fig. 1. An example of hypervisors, virtual machines and physical machines on the cloud.

2.1. Virtualization

Virtualization technologies provide an abstraction of computing resources to the end user [15,16]. From the resources point of view, virtualization provides a way to create logical subsets of resources from a physical machine. From the users point of view the virtual environment behaves like a real physical machine (PM). The flexibility of virtualization is why it is a key component in cloud computing.

On the cloud, physical machines are logically partitioned using virtual machine techniques such as Xen [17], VMWare [18] and KVM [19]. Virtual machines are a way to provide computing resources to users, which they can manage and configure to fit their specific needs.

For example in Fig. 1, there are two physical machines, both hosting three virtual machines managed by a hypervisor. From the provider's perspective, they have two physical machines to maintain, while from the users perspective it appears that they have access to six different machines.

2.2. MapReduce

Currently, the only viable way to work with voluminous amounts of data is to use a divide and conquer strategy. The essential principle to the divide and conquer strategy is to break down a large problem into several smaller problems. MapReduce achieves this by dividing the storage and processing of data and distributing them amongst a number of computers, known as nodes, in a network. In other words, MapReduce is a framework for writing distributed applications that process large amounts of data.

Traditionally, parallel or distributed environments required programmers to handle a multitude of issues. For instance, how does one divide the large problem? Exactly, what sub-problems and what tasks need to be performed? How will the different tasks be instantiated? How will they communicate? How will the workload be distributed? What happens with the results produced by the individual tasks? What happens if there is a software error? What happens if there is a hardware failure? How does one coordinate the system so that one can avoid deadlock or a race condition? These myriad details are quandaries that many developers have had to contend with.

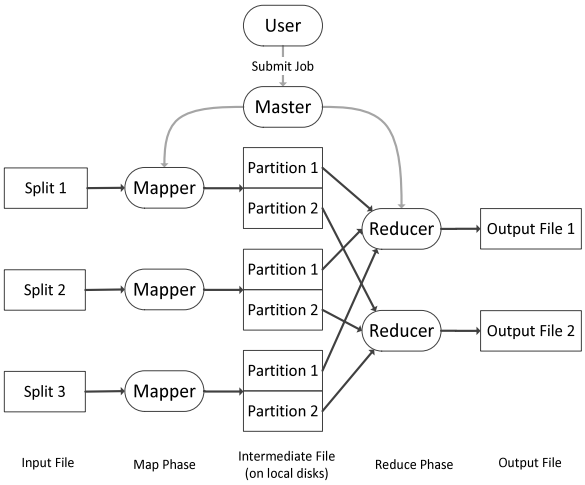

Fig. 2. MapReduce dataflow.

The MapReduce framework was proposed as a way to alleviate the burden placed on software developers and to prevent developers from having to reinvent solutions to similar or even identical problems. In essence, MapReduce provides an abstraction that allows software developers to ignore the details and allows them to focus their attention on the problem they are really trying to solve. This abstraction is one of the main advantages of MapReduce. While language extensions such as OpenMP [20] or MPI [21] provide parallel systems a way to communicate, they still require developers to keep track of resources and provide little support when handling large amounts of data. From the programmers perspective MapReduce is a relatively easy way to create distributed applications compared to traditional methods. It is one of the main reasons for MapReduce's popularity.

The idea of MapReduce evolved from principles used in functional programming. The basic principles of MapReduce can be summarized as being two distinct programming functions, a map function and a reduce function. It is the responsibility of the programmer to provide these functions. These functions are then incorporated into two tasks known as a mapper and reducer, which handle the map and reduce functions respectively. In this paper, the terms mapper and reducer are used interchangeably with the terms map task and reduce task.

The dataflow for the MapReduce framework is shown in Fig. 2. In Hadoop, additional tasks and nodes are required to handle errors, initiate jobs, and to coordinate the various tasks. For the sake of simplicity, details relevant to coordination and communication as well as other extraneous details used have been simplified, and amalgamated into a single master task.

In MapReduce, the purpose of the map and reduce functions is to handle sets of keys–value pairs. When a user runs a MapReduce program, data from a file or set of files is split amongst the mappers provided and read as a series of key–value pairs. The mapper then applies the map function on these key–value pairs. It is the duty of the map function to derive meaning from the input, to manipulate or filter the data, and to compute a list of key–value pairs. The list of key–value pairs is then partitioned based on the key, typically via a hash function (1). During this process, data is stored locally in temporary intermediate file.

$$\text{partitionNumber} = \text{key} \,\%\, \text{totalNumberOfReducers}. \tag{1}$$

In MapReduce, the purpose of partitioning is to ensure that each mapper sends all its key–value pairs with the same key to the same reducer. Eventually, all of the key–value pairs for a particular partition merge at a specific reducer. During the merge, all keys are sorted into a unique list of keys with a corresponding list of values for each of these keys. The reducer then executes in a loop a reduce function which takes as input a key and a list of values. Once the reduce function finishes computing the data an output file is produced. Each reducer generates a separate output file. These files can be searched, merged or handled however the user wants once all reducers have completed their workload.

3. Proposed techniques and implementation

Hadoop has garnered much popularity in both academia and industry. The popularity it enjoys is a result of both its speed and economic efficacy. Furthermore, since Hadoop is open source, academics have been provided a useful platform on which to base their research. However, one of the drawbacks of the Hadoop implementation is it assumes that the computing nodes in the network are homogeneous [22].

Consequently, Hadoop exhibits several inefficient behaviors when employed in a heterogeneous environment. This paper therefore proposes some methods that can improve the performance of MapReduce when executing in such an environment.

The research model for this study is presented in Fig. 3, which shows a network that consists of several physical machines. Each physical machine (PM) has a limited number of virtual machines (VM). Without losing generality, virtual machines are used as a basic unit with which to execute a task. Each virtual machine represents a processing unit that can run either a map task or a reduce task. Due to the heterogeneous nature of the environment, the processing capabilities of any particular virtual machine may differ from other virtual machines in the environment.

3.1. Dynamic data partitioning

A file system that handles storing and the manipulation of the file or files in which the dataset is stored across multiple machines is called a distributed file system. In the age of big data, the size of a dataset can exceed the storage capacity of a single physical machine. It therefore becomes necessary to distribute the storage of the dataset on multiple machines. For this reason Hadoop has its own distributed file system called the Hadoop Distributed File System (HDFS).

In Hadoop, a MapReduce job begins by first reading a large input file. This file is usually stored on the HDFS. The data from this file is then divided into a set of fixed sized pieces known as splits. Hadoop then creates a mapper for each split. Hadoop assumes a homogeneous cluster in which each node has the same processing power and capabilities. If this is the case, then each mapper will finish processing its split at approximately the same time. Of course in a heterogeneous network, where nodes do not have this property some nodes will complete their work sooner than others will.

If a mapper with greater processing efficiency is able to reduce the workload from one that is slower one, then the maximum time to complete the task for those two mappers is reduced. It would seem then that it would be desirable to have all mappers dynamically sharing their workload with one another. Unfortunately, access data rates between nodes on the HDFS are inconsistent due to issues of data locality. Therefore, we propose the following dynamic data partitioning scheme in which each node performs dynamic data partitioning irrespective of other nodes on the network.

An example of the dynamic data partitioner is shown in Fig. 4. In this example, a 600 GB file is used as input data. In this scenario, the data is to be divided up into six equal sized pieces, which are to sent to six virtual machines. Each of these virtual machines execute a map task. Each virtual machine is given a value n that indicates the relative processing ability of each machines virtual

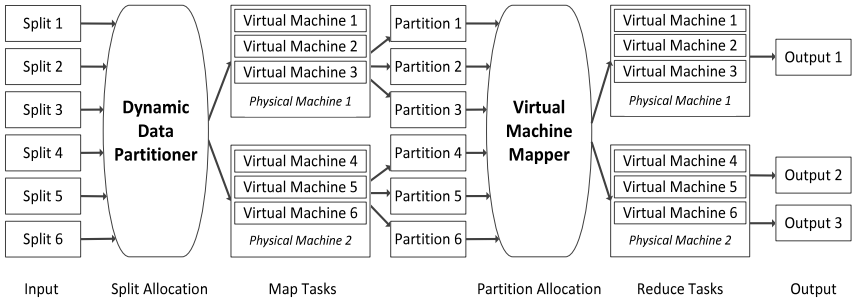

Fig. 3. Dataflow of dynamic data partitioner and virtual machine mapper.

processing unit (VPU). Both the CPU of the physical machine and the total number of vCPU assigned to the VM determines the actual processing ability of the VPU.

For instance, the virtual machine VM1 has an n value of 10 and the virtual machine VM2 has an n value of 2. This means that VM1 is able to process data 5 times faster than VM2. The processing speed of each virtual machine is calculated prior to execution using a profiling tool.

As previously mentioned, the proportion of data to be reassigned amongst virtual machines is determined by the processing ability of all the virtual machines running on the same physical machine. The following algorithm calculates the amount of data to be assigned to each virtual machine:

Algorithm 1 Data Repartitioning

Input:
SPM: set of all physical machines
PM: physical machine
VM: virtual machine

```
1.  for each PM on SPM
2.    //calculate fragment size
3.    for each VM on PM
4.      totalDataSize = totalDataSize + VM.init.splitSize
5.      totalSpeed = totalSpeed + VM.processingSpeed
6.    end for
7.    fragmentSize = totalDataSize/totalSpeed
8.    //calculate data to be reassigned to each VM
9.  for each VM on PM
10.       VM.ideal.splitSize = VM.processingSpeed *
                               fragmentSize
11.
      VM.diff.splitSize = VM.init.splitSize-VM.ideal.splitSize
12.   end for
13.  // data migration from over-loading VM_o to
     under-loading VM_u in O(n), where n is the number of VMs
14.   for each over-loading VM, e.g. VM_o
15.     for each under-loading VM, e.g., VM_u
16.       Send.fragment (fragment.size =
           min{VM_o.diff.splitSize, VM_u.diff.splitSize},
          VM_o, VM_u)
17.       VM_o.diff.splitSize − = fragment.size
18.       VM_u.diff.splitSize + = fragment.size
19.     end for
20.   end for
21.  end for
```

For example, in Fig. 4 there are two physical machines. Both physical machines have three virtual machines. The attributes of which are summarized in Table 1.

Once the initial input splits are designated to each virtual machine the DDP repartitions the data on each physical machine.

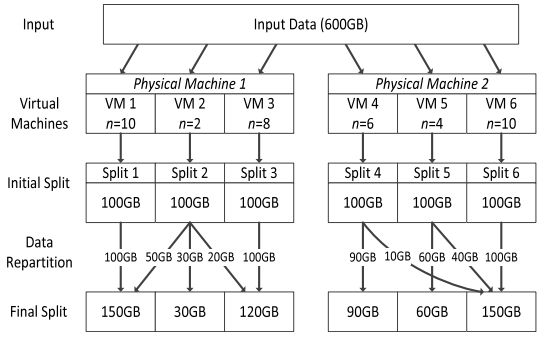

Fig. 4. Dynamic data partitioner.

Table 1
Summary of initial inputsplit.

Physical machine	Virtual machine	VPU speed	Input data (GB)
1	1	10	100
	2	2	100
	3	8	100
	4	6	100
2	5	4	100
	6	10	100

On PM1 there is three virtual machines VM1, VM2 and VM3. VM1 has a VPU speed of 10, VM2 has a VPU speed of 2 and VM3 has a VPU speed of 8. Each virtual machine has an initial split size of 100 GB. Consequently, the total data size of the three virtual machines is 300 GB, and the total VPU speed of the three virtual machines is 20 units. The input split is then divided into fragments. The size of fragment is calculated using the following equation:

$$\text{fragmentSize} = \text{totalDataSize}/\text{totalSpeed}. \quad (2)$$

The split size for each virtual machine is then reassessed by multiplying the VPU speed of each virtual machine by the fragmentSize. For PM1 the fragmentSize is 300/20 = 15.

$$\text{splitSize} = \text{VPU.processingSpeed} * \text{fragmentSize}. \quad (3)$$

Therefore, for PM1, where VM1 has a VPU speed of 10, its split size is reassessed to be 10 * 15 = 150 GB. All other data splits are calculated using the same method. The results are summarized in Table 2.

3.2. Virtual machine mapping

After a mapper processes an input split, the mapper writes to a temporary file on the local disk. The mapper output is a list of key–value pairs. This list is partitioned based on the key–value, and on how many reducers there are. Each partition is then sent to a different reducer. The reducer receives and merges this data from each of the mappers. The reducer then processes this data and produces the final output.

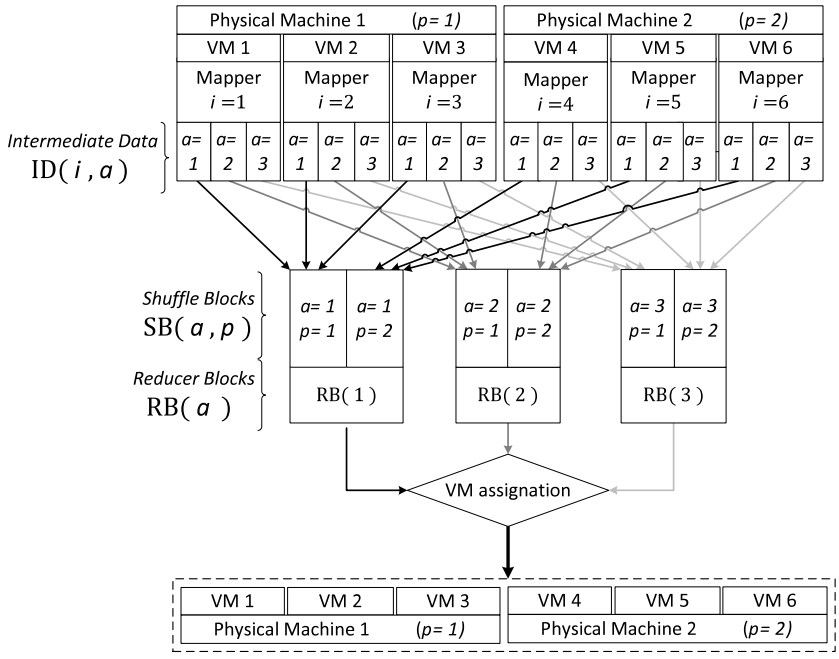

Fig. 5. Virtual machine mapper model.

Table 2
Summary of final inputsplit.

Physical machine	Virtual machine	VPU speed	Input data (GB)
	1	10	150
1	2	2	30
	3	8	120
	4	6	90
2	5	4	60
	6	10	150

A master node determines where mappers or reducers reside on a network. When assigning a mapper to a node, it is important that it is located on or near the data it is going to access. This is because transferring data from one node to another takes time and delays task execution. The problem of determining where to place a task on the network so that it is close to the data it uses is known as *data locality*. Data locality is a key concept in MapReduce and has a major impact on its performance.

A key concept for MapReduce is that: "moving computation towards data is cheaper than moving data towards computation". Even though Hadoop uses this concept when determining where to execute its mappers, it does not exploit this concept for its reducers and locate reducers near these partitions. We therefore propose for this purpose a virtual machine mapper (VMM) which allocates the reducer to the appropriate physical machine based on partition size and the availability of a virtual machine (see Fig. 5).

Given a mapper i, the intermediate data that resides on that mapper's ath partition can be described as ID(i, a). Before physical sending this data to a VM, the partition data from all the mappers forms a set of shuffle blocks. Each shuffle block contains all the data produced by the mappers on a single physical machine for a specific reducer. The total number of shuffle blocks (NSB) destined to each reducer is equal to the total number of reducers (N) multiplied by the total number of physical machines (P).

$$\mathrm{NSB} = \mathrm{N} * \mathrm{P}. \tag{4}$$

A shuffle block $SB\,(a, p)$ is created by combining partition a data from each mapper on a given physical machine p. Given P physical machines, with M mappers on each machine, this can be summarized using the following formula:

Where $\mathrm{ID}(i, a)$ = intermediate data stored in the ath partition on mapper i
Where $\mathrm{SB}(a, p)$ = shuffled block of the ath partition gathered from physical machine p
Where $\alpha = (p-1) * \mathrm{M} + 1$
Where $\beta = (p-1) * \mathrm{M} + \mathrm{M}$
$SB\,(a, p) = \sum_{i=0}^{M} ID(i, a)$

$$\sum_{\alpha \le i \le \alpha+\beta} ID\,(i, a)\,. \tag{5}$$

For instance based on an environment that has two physical machines, with three virtual machines on each virtual machine:

$\mathrm{SB}(1,1) = \mathrm{ID}(1,1) + \mathrm{ID}(2,1) + \mathrm{ID}(3,1)$.
$\mathrm{SB}(1,2) = \mathrm{ID}(4,1) + \mathrm{ID}(5,1) + \mathrm{ID}(6,1)$.
$\mathrm{SB}(2,1) = \mathrm{ID}(1,2) + \mathrm{ID}(2,2) + \mathrm{ID}(3,2)$.
$\mathrm{SB}(2,2) = \mathrm{ID}(4,2) + \mathrm{ID}(5,2) + \mathrm{ID}(6,2)$.

Shuffle blocks that have the same value of a form a single reducer block (RB). A reducer block is a superset of data and contains all the data to be processed by a single reducer.

$$\mathrm{RB}(a) = \sum_{j=(i*M)+a}^{(P*M)} SB(i, a). \tag{6}$$

For instance based on an environment that has two physical machines, with three virtual machines on each virtual machine and the shuffle blocks above:

$\mathrm{RB}(1) = \mathrm{SB}(1,1) + \mathrm{SB}(1,2)$.
$\mathrm{RB}(2) = \mathrm{SB}(2,1) + \mathrm{SB}(2,2)$.

Once reducer blocks are formed, the VMM assigns them to an appropriate mapper based on data locality and is based on first come first served (FCFS) and BestFit and are assigned to a VM on a PM based on the aggregate partition size of its component shuffle blocks. For example if the aggregate partition size of $\mathrm{SB}(1, 1) > \mathrm{SB}(1, 2)$ then RB(1) would be assigned to physical machine 1. However if $\mathrm{SB}(1, 1) < \mathrm{SB}(1, 2)$ then RB(1) would be assigned to physical machine 2.

Fig. 6 shows an example of the VMM ascribing reduce tasks to physical machines. In this example, there are six mappers on two physical machines, with three mappers per physical machine. Since this job requests three reduce tasks, each mapper creates three partitions. The total amount of data to be received by each reducer is then deduced by summing up the respective partition at each mapper.

Based on the concept of data locality, reducers are assigned locations on physical machines based on where the data for each reducer is stored. On a physical machine there are multiple virtual machines. Once a reducer is assigned to a physical machine the reducer is then assigned whichever virtual machine has the fastest processing speed. The algorithm for the virtual machine mapper is presented as follows:

Algorithm 2 Virtual Machine Mapper

Input:

SPM:	set of physical machines
PM:	physical machine
SVM:	set of virtual machines
VM:	virtual machine
i	reducer index
j	partition index

```
1.  // retrieve partition metadata
2.  SPM = the set of all physical machine on the network
3.  reducers = array of all reducers
4.  for each PM on SPM
5.    for eachmapper on PM
6.      fori = 1to NumberOfReducers
7.        forj = 1to NumberOfPartitions
8.          reducers[i].partition[j].size =
            mapper.partition[i].size
9.          reducers[i].partitionSize + =
            mapper.partition[i].size
10.       end for
11.     end for
12.   end for
13.   // Perform BestFit reducer assignment
14.   // reducer is assigned whichever virtual machine has the
        fastest processing speed.
15.   // assign reducers to physical machines based on
        reducers[i].partition[j].size and available VM's
16.   for each PM on SPM
17.     SVM = the set of all virtual machines on the PM
18.     reducers = array of reducers on PM
19.     // sort virtual machines based on speed.
20.     sortByProcessingSpeed(SVM)
21.     fori = 1to PM.numberOfReducers
22.       SVM[i] = reducers[i]
23.     end for
24.   end for
```

For example, in Fig. 6 there are two physical machines. Both physical machines have three virtual machines each running a mapper. Each mapper produces three data partitions which are to be assigned to three reducers. Using Algorithm 2, reducer 1 is assigned to physical machine 1, virtual machine 1. This is because physical machine 1 stores the largest fraction of the data designated for that reducer. Similarly, reducer 2 and reducer 3 are assigned to a virtual machine on physical machine 2. On physical machine 1 there are three viable virtual machines (VM1, VM2, VM3). Reducer 1 is assigned to VM1 as it has the fastest processing speed. On physical machine 2 there are also three viable virtual machines (VM4, VM5, VM6) each with different processing speeds.

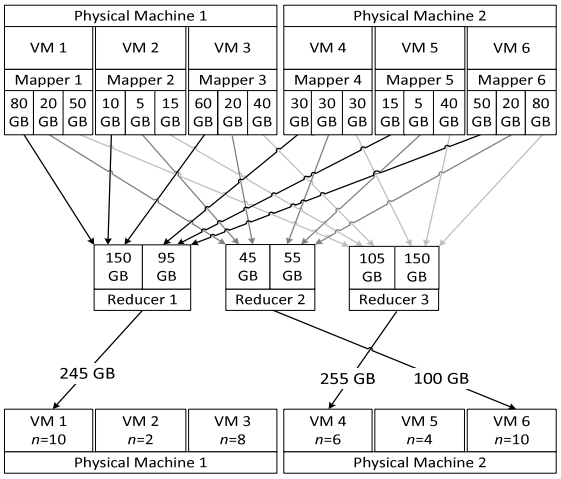

Fig. 6. Virtual machine mapper.

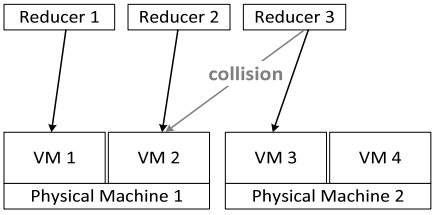

Fig. 7. When more reducers request the same physical machine than there are virtual machines available, the VMM has to select a virtual machine on a different physical machine.

Table 3
Virtual machine mapping.

PM	VM	VPU speed	Data locality (PM)	Data size (GB)	Total data size (GB)	Reducer
1	1	10	1 2	150 95	245	1
	2	2	NIL	0	0	n/a
	3	8	NIL	0	0	n/a
2	4	6	1 2	105 150	255	3
	5	4	NIL	0	0	n/a
	6	10	1 2	45 55	100	2

Since Reducer 2 is processed first it is assigned to the fastest virtual machine (VM6). Consequently, reducer 3 is assigned to the second fastest virtual machine (VM4). Details regarding the virtual machine mapping for Fig. 6 are summarized in Table 3.

In this example, all of the reducers are allocated to the appropriate physical machine. However, there may be cases when there are no virtual machines available. In this case, the reducer has to be allocated to another physical machine. This is achieved using the best fit selection method.

When a single physical machine has more reducers requesting a virtual machine than there are available virtual machines, the VMM has to locate the reducer on another physical machine. However, arbitrarily rejecting reducers based on a first come first served (FCFS) order is suboptimal (see Fig. 7).

3.3. Priority based virtual machine mapping

To improve data locality further, the VMM assesses the size of the data contributed by each physical machine to each reducer. The VMM then gives priority to the reducers with the

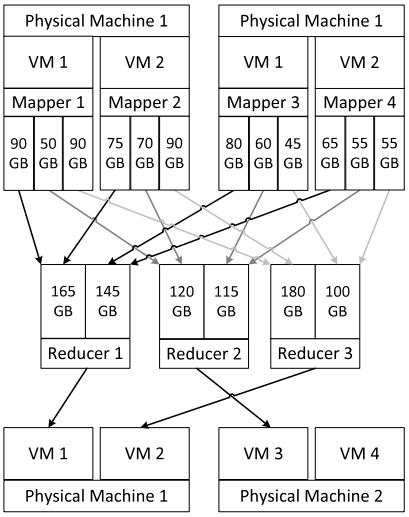

Fig. 8. VMM priority fit: reducers are assigned to a virtual machine based on data locality. Using a priority based order, Reducer 2 is reassigned to physical machine 2.

Table 4
Priority based order.

Reducer	Data locality (PM)	Data size (GB)	Absolute difference	Priority
1	1 2	165 145	20	2
2	1 2	120 115	5	3
3	1 2	180 100	40	1

greatest difference between the largest and second largest data contributions.

Fig. 8, gives an example of the VMM using priority-based reassignment. Using the laws of data locality, all three reducers should be assigned to PM1. This is not possible, as there are only two virtual machines available. Priority based order is thus used to determine which reducers should be assigned to PM1 and which to reject. An example of the priority-based calculations used for Fig. 8, is presented in Table 4.

By prioritizing reducers, one can improve data locality. This is because priority is given to those reducers that receive data disproportionally more from a single physical machine. This is preferable to FCFS reducer order (e.g. reducer 1, reducer 2, …, reducer n) which does not use such a criteria to differentiate between reducers.

3.4. Load aware virtual machine mapping

One of the original drawbacks of VMM is that it treats all reducers equally regardless of the difference in the workload. This method works in a homogeneous system, in which all virtual machines have the same characteristics. However, in a heterogeneous environment one needs to consider the VPU of each virtual machine. Therefore, this paper proposes Load Aware Virtual Machine Mapping (LA-VMM) which takes into account the processing speed of each virtual machine. LA-VMM achieves this by assigning reducers with greater workloads to virtual machines with faster processing speeds.

In Fig. 9, the LA-VMM assigns reducer 1 to PM1, and reducer 2 and reducer 3 to PM2. On PM1, reducer 1 is assigned to VM1 as it has the fastest processing speed. On PM2 there are also three viable virtual machines (VM4, VM5, VM6) each with different processing

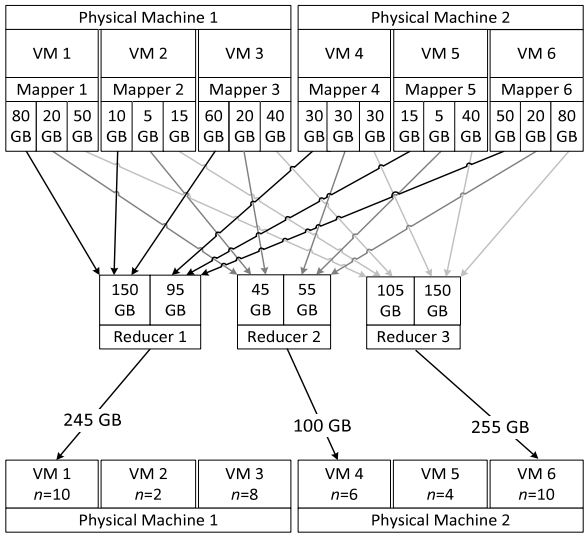

Fig. 9. Load aware virtual machine mapper.

Table 5
Load aware—virtual machine mapping.

PM	VM	VPU speed	Data locality (PM)	Data size (GB)	Total data size (GB)	Reducer
1	1	10	1 2	150 95	245	1
	2	2	NIL	0	0	n/a
	3	8	NIL	0	0	n/a
	4	6	1 2	45 55	100	2
2	5	4	NIL	0	0	n/a
	6	10	1 2	105 150	255	3

speeds. Since Reducer 2 has the greatest workload it is assigned to the virtual machine which has the fastest processing speed (VM6). Consequently, reducer 3 is assigned to the second fastest virtual machine (VM4). Details regarding the virtual machine mapping for load aware dynamic data partitioning LA-VMM are summarized in Table 5.

4. Evaluation

To evaluate the performance of the proposed techniques, we have implemented the dynamic data partitioning and virtual machine mapping algorithms and tested these methodologies with a 900 MB randomly generated input data file on a simulated MapReduce environment. The performance analysis was reported with synthesized datasets, where the number of key–value pairs was set equal to the number of reducers. A 1:3 performance ratio (low heterogeneity) and 1:5 performance ratio (high heterogeneity) indicate the possible difference in processing ability between any two virtual machines in the network.

4.1. Data locality

We measure data locality by the total amount of data stored locally on the physical machine for each virtual machine. When data is local, access is fast. Data locality is therefore a reflection of performance for MapReduce. The heterogeneous environment is tested using either 2 or 5 virtual machines per physical machine. The total number of virtual machines used was 10, 20 or 30 virtual machines.

Fig. 10 depicts data locality in a heterogeneous environment. Fig. 10(a) has a heterogeneous performance ratio of 1:3, with 2

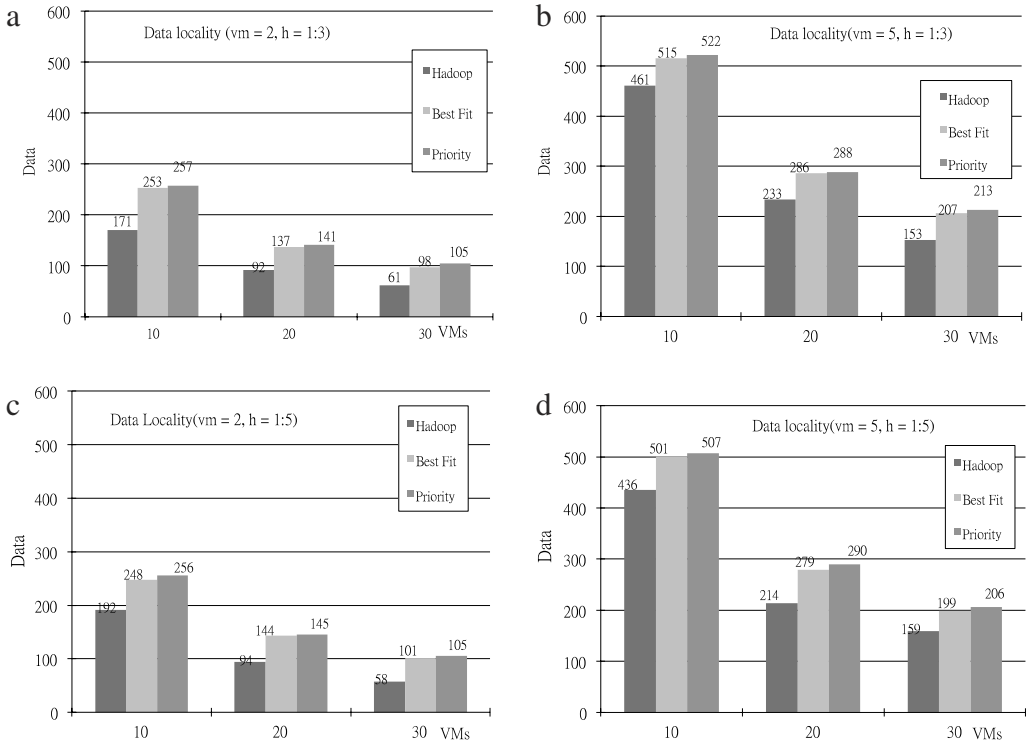

Fig. 10. Data locality (a) low heterogeneity, vm = 2, (b) low heterogeneity, vm = 5, (c) high heterogeneity, vm = 2, (d) high heterogeneity, vm = 5.

virtual machines per physical machine. One can see that both best fit and priority based VMM improved data locality compared to Hadoop by more than 33%. Fig. 10(b) has 5 virtual machines per physical machine. In this case the difference in data locality between best fit and priority methods and Hadoop has reduced. This is due to increased ratio of data shared between virtual machines on the same physical machine. Despite the increase to a heterogeneous performance ratio of 1:5, data locality in Fig. 10(c) and (d) mostly mirror the performance of Fig. 10(a) and (b).

4.2. Task execution time

Experiments in this section explore how long Mappers and Reducers take to complete their tasks. The heterogeneous environment is tested using either 2 or 5 virtual machines per physical machine. The total number of virtual machines used was 10, 20 or 30 virtual machines.

In Fig. 11, we explored map completion time. In the Map phase, the Dynamic Data Partitioner repartitions the input split size based on virtual machine performance. Map completion time reduced, by a factor of 44%, when the number of virtual machines per physical machine increased. This highlights how inefficient Hadoop's implementation is in a heterogeneous environment.

In Fig. 12, we explored reduce completion time. In the Reduce phase, the Virtual Machine Mapper selects which virtual machine a reducer should reside. In this paper, we presented two ways for the VMM to decide which order reducers are processed, which we designated best fit and priority based order. Fig. 12(a) shows an improvement in reduce completion time of 14%, Fig. 12(b) shows an improvement in reduce completion time of 16%, Fig. 12(c) shows an improvement in reduce completion time of 23%, and Fig. 12(d) shows an improvement in reduce completion time of 29%. These simulation results show that our proposed methods save more time as heterogeneity increases.

In Fig. 13, we explored the total completion time. Total completion time combines the time taken for both reduce and map completion times. In Fig. 13, it is shown that the combination of DDP and VMM has shortened total completion time by up to 29% for low heterogeneous environments and shortened total completion time by upto 41% in high heterogeneous environments.

In Fig. 14, we explored the total completion time using load aware optimization. The heterogeneous environment used 3 physical machines with 5 virtual machines per physical machine. Therefore, a total of 15 virtual machines were used. The test was performed with a randomly generated 900 GB file. The load awareness optimization is an optimization of the Virtual Machine Mapper which focuses on placing reducers with heavier workloads on to faster virtual machines, consequently the map phase shows no improvement. The load aware optimization improved the reduce phase by 13% for both best fit order and priority based order Virtual Machine Mapping.

In Fig. 15, we explored the total completion time for MapReduce using different number of virtual machines per physical machine. The heterogeneous environment used a fixed number of 12 workers. The test was performed with a randomly generated 600 GB file. The purpose of the test was to determine how differing ratio of physical machines and virtual machines affects MapReduce completion time. When the number of virtual machines increased on each physical machine increased, there were more input splits presented to the dynamic data mapper on each physical machine. This provides the data repartitioner access to more locations with which to distribute the data. This provides greater opportunities for more even distribution amongst the virtual machines, if the performance improves. Furthermore, the virtual machine mapper performance also improves as having more virtual machines reduces chances of collisions as well as increased data locality.

In Fig. 16, we compared the total completion time for a MapReduce using different data sizes. The heterogeneous environment used 3 physical machines with 5 virtual machines per physical machine. The amount of data used as input for MapReduce ranged from 400 GB to 1200 GB. The results show that the time to completion increases for all methods as the amount of

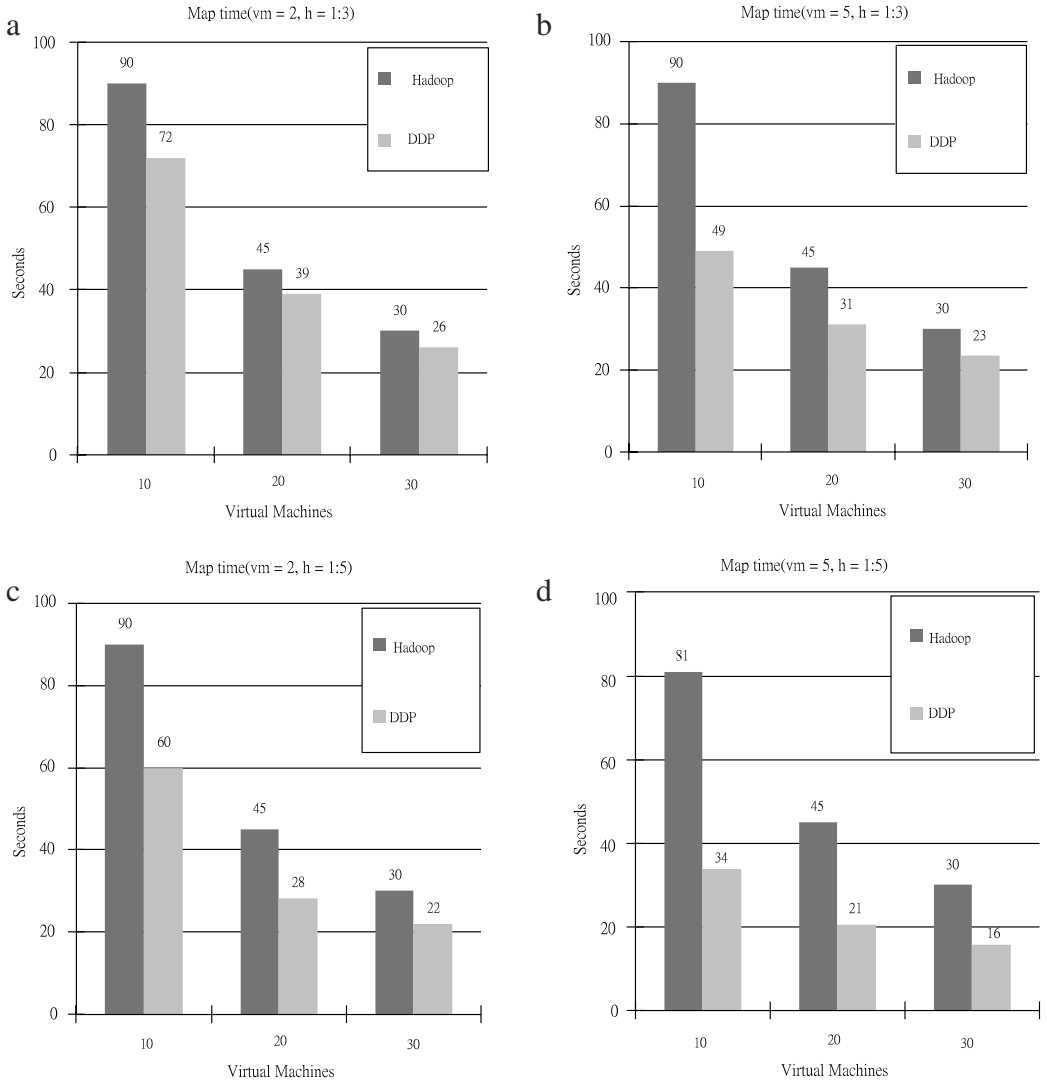

Fig. 11. Map completion time (a) low heterogeneity, vm = 2, (b) low heterogeneity, vm = 5, (c) high heterogeneity, vm = 2, (d) high heterogeneity, vm = 5.

data to process increases. However, the dynamic data partitioning with best fit or priority based virtual machine mapping method performed better than Hadoop as the data load increased.

5. Related work

There has been a variety of studies done by researchers on MapReduce and heterogeneous environments. We begin by summarizing some of the recent works here.

MapReduce frameworks such as Hadoop make an assumption that they will be deployed in a homogeneous environment. However, there are times where that assumption is untrue. Khalil, Salem, Nassar, and Saad [23] identified three reasons why this assumption may be broken. Firstly, it is often impossible or even undesirable to have only one type of machine in a network. Secondly, homogeneity is not satisfied by virtualized data centers. Thirdly, scheduling decisions do not take into account differences in workload characteristics for different jobs.

Overall, researchers [22–26] have noted that various aspects of MapReduce's performance is not as efficient as it could be within a heterogeneous environment. There has been prior research into improving data locality within the MapReduce environment. As has been noted amongst the literature [4,27,28], data locality has a large influence on MapReduce performance. From the mappers perspective, researchers have studied how throughput of MapReduce can be optimized during periods of heavy traffic. Meanwhile developers of a locality aware reduce task scheduler (LARTS) [28], noted that Hadoop did not schedule reducers using any data locality optimization. The LARTS study focused on reducing network traffic by identifying whether partition data was on the same node, on the same rack or on a different rack when scheduling a reducer. While LARTS was shown to outperform Hadoop reduce task scheduler for homogeneous environments, it had no provisions for heterogeneous environments.

Another method to improve data locality in heterogeneous environment is to focus on data placement [22,25]. In this scenario, a large dataset is partitioned into data fragments and then distributed across multiple heterogeneous nodes. The intent is to ensure each node has a balanced processing load. For this purpose, the researchers implemented a data reorganization and data redistribution method within the HDFS. The purpose of this method is to distribute more fragments near higher performing nodes. As MapReduce executes on top of HDFS this method is orthogonal to the one proposed in this paper which works above the file system layer.

Research into on MapReduce clouds [26] looked at how locality aware VM reconfiguration could be attained. The researchers took a different approach to the problem of data locality, and modified the computing environment based on the locality of the data.

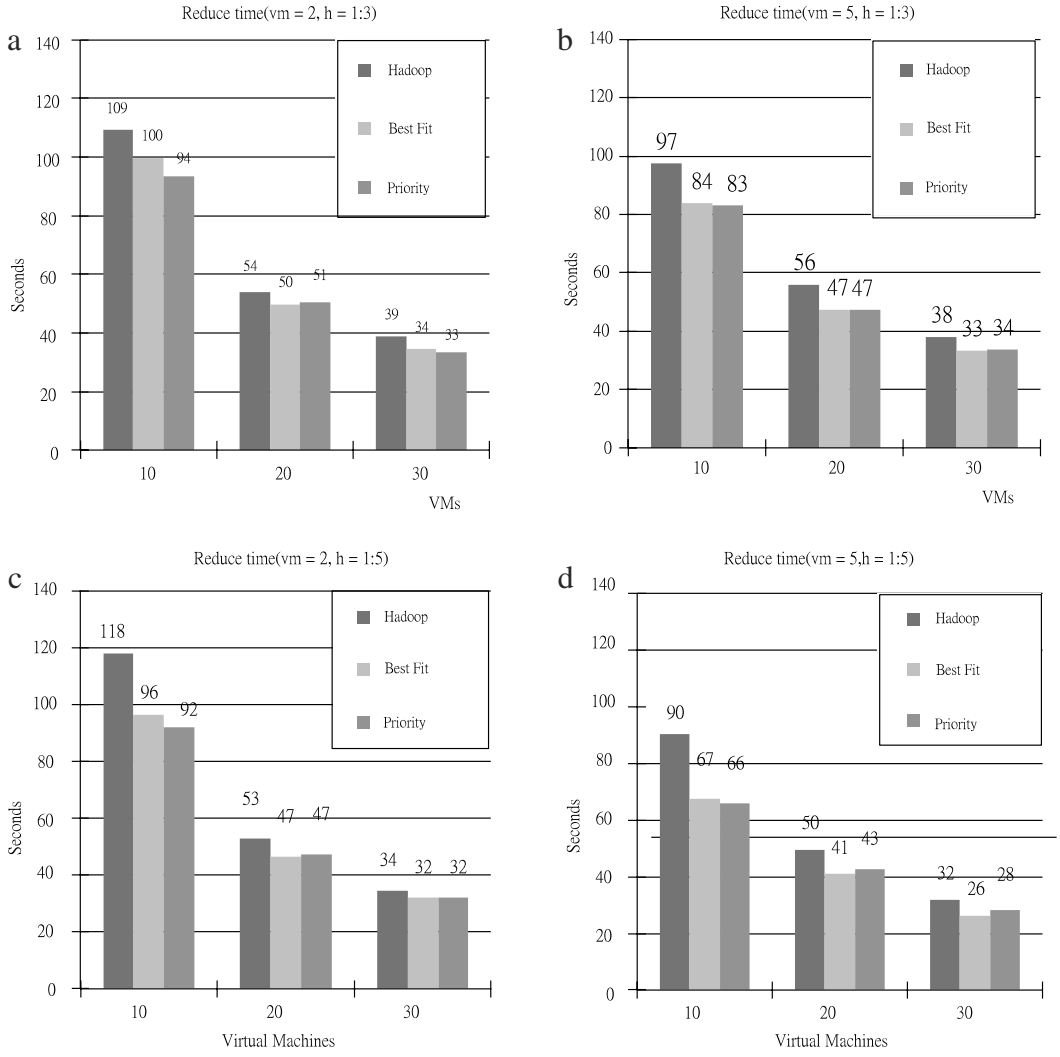

Fig. 12. Reduce completion time (a) low heterogeneity, vm = 2, (b) low heterogeneity, vm = 5, (c) high heterogeneity, vm = 2, (d) high heterogeneity, vm = 5.

Essentially, the researchers adjusted the computing capabilities of individual VM's in order to maximize utilization of resources.

Li et al. designed topology aware scheduling algorithms for map tasks [29]. Extending the rack aware feature of the existing scheduler, this work makes MapReduce scheduler aware of network topology to provide one more level of caching. In such design, a master node has information about the physical location of the requesting client node to enhance resource provisioning. As data locality is critical to map performance, many work have been done to preserve locality via map scheduling or input replication. Another prior study improved the locality of map and reduce tasks in the cloud by locality-aware VM placement [30]. This study proposes data and virtual machine placement strategies accord to the cost of data shuffling between map tasks and reduce tasks. The authors assumed that having prior knowledge about the characteristics of MapReduce workloads before customers execute them, the cloud scheduler could place data to the proper physical machines using the workload information. According to data layout, the VM scheduler places VMs to the physical systems with the corresponding input data.

One of the benefits of MapReduce is that it is fault tolerant, that it handles these faults automatically and that it keeps its fault tolerant implementation hidden from the programmer. However, when Hadoop is deployed in a heterogeneous environment, certain assumptions are false. Consequently, faults maybe detected falsely, or it may respond with an under correction or overcorrection. One of the methods employed by MapReduce to handle faults is known as speculative execution. Speculative execution is a method that detects a slow task, and then duplicates that task on another node. A task can be slow for many reasons. For instance the node the task is running on may underperform due to a hardware fault or due to misconfiguration. How to determine which node is slow and when to trigger speculative execution has been a subject of several papers [31–34].

6. Conclusion

This paper is based on MapReduce and the Hadoop framework. Its purpose is to improve the performance of MapReduce distributed application when executing in a heterogeneous environment. By dynamically partitioning input data being read by map tasks and by judicious assignation of reduce tasks based on data locality using a Virtual Machine Mapper. Furthermore, this paper presents an optimization of this method called a Load Aware Virtual Machine Mapper.

Simulation and experimental results show an improvement in MapReduce performance, improving data locality by 33% and optimizing total completion time by 41%. Furthermore, using the Load Aware Virtual Machine Mapper there was additional 13% improvement in reduce phase completion time.

In future research, this work can be expanded to dynamically determine the number of reducers deployed on the MapReduce

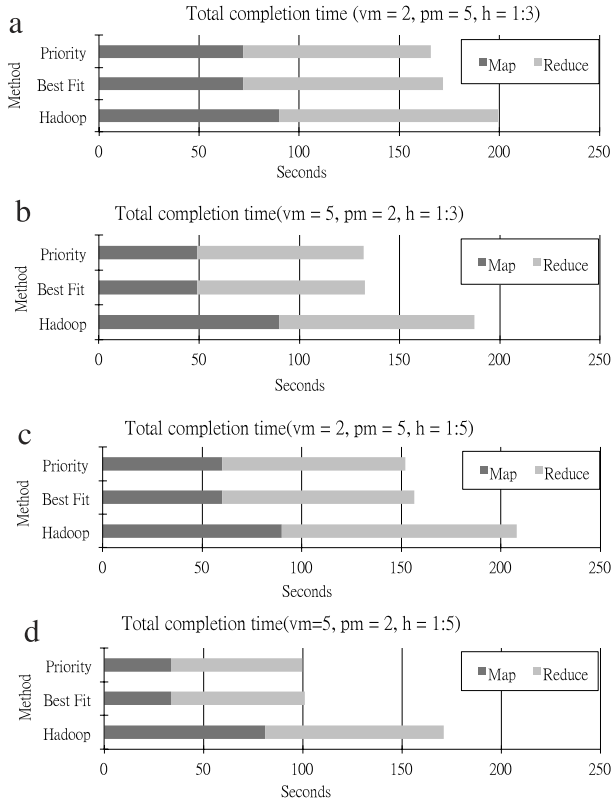

Fig. 13. Total completion time (a) low heterogeneity, vm = 2, (b) low heterogeneity, vm = 5, (c) high heterogeneity, vm = 2, (d) high heterogeneity, vm = 5.

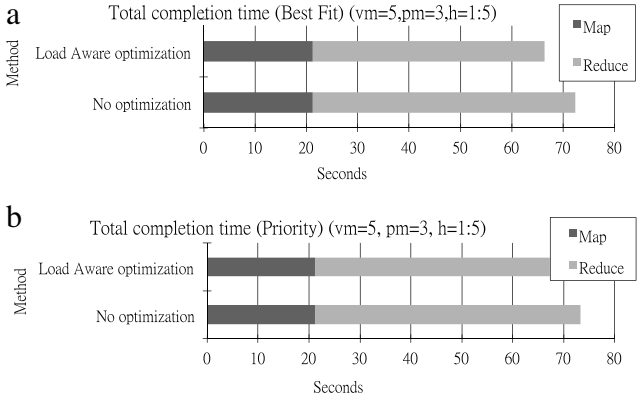

Fig. 14. Total completion time with load aware optimization: (a) Best fit method. (b) Priority method.

environment. This is an important topic, which analyzes the cost–benefits of increasing the number of reducers, and compares whether the impact on performance justifies the amount of computing resources used. Furthermore, this study has showed some encouraging results and it is hoped that this work can be ported from a simulated environment to a physical platform.

Acknowledgments

We thank the anonymous reviewers for their insightful comments. This work was funded by Ministry of Science and Technology, Taiwan, under grant number NSC-101-2918-I-216-001.

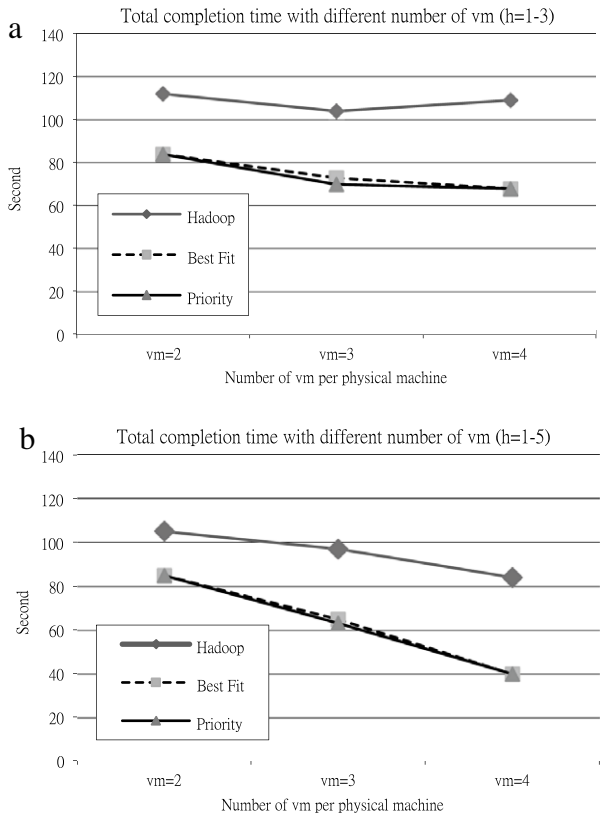

Fig. 15. Total completion time for different number of virtual machines. Using high heterogeneity, vm = 5.

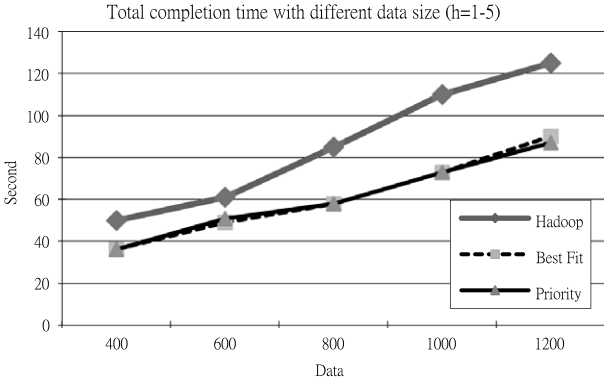

Fig. 16. Total completion time for different number of virtual machines. Using high heterogeneity, vm = 5.

References

[1] A.B. Patel, M. Birla, U. Nair, Addressing big data problem using Hadoop and Map Reduce, in: 2012 Nirma University International Conference on Engineering, NUiCONE, 2012, pp. 1–5.
[2] B. Feldman, E.M. Martin, T. Skotnes, Big data in healthcare hype and hope, 2012.
[3] R. Smolan, J. Erwitt, The Human Face of Big Data, Against All Odds Productions, 2012.
[4] T. White, Hadoop: The Definitive Guide, third ed., O'Reilly Media, Inc., 2012.
[5] K.-H. Lee, Y.-J. Lee, H. Choi, Y.D. Chung, B. Moon, Parallel data processing with MapReduce: a survey, ACM SIGMOD Rec. 40 (2012) 11–20.
[6] Apache Hadoop. Available at: http://hadoop.apache.org [August 12, 2013].
[7] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Commun. ACM 51 (2008) 107–113.
[8] S. Ghemawat, H. Gobioff, S.-T. Leung, The Google file system, ACM SIGOPS Oper. Syst. Rev. (2003) 29–43.
[9] J. Dittrich, J.-A. Quiané-Ruiz, Efficient big data processing in Hadoop MapReduce, Proc. VLDB Endow. 5 (2012) 2014–2015.
[10] N. Sultan, S. van de Bunt-Kokhuis, Organisational culture and cloud computing: coping with a disruptive innovation, Technol. Anal. Strateg. Manag. 24 (2012) 167–179.

[11] J. Lin, C. Dyer, Data-intensive text processing with MapReduce, in: G. Hirst (Ed.), Synthesis Lectures on Human Language Technologies, Morgan & Claypool, 2010, pp. 1–35.
[12] L. Wang, G. Von Laszewski, A. Younge, X. He, M. Kunze, J. Tao, C. Fu, Cloud computing: a perspective study, New Gener. Comput. 28 (2010) 137–146.
[13] M. Böhm, S. Leimeister, C. Riedl, H. Krcmar, Cloud computing and computing evolution, Technische Universität München, TUM, Germany, 2010. pp. 3–6.
[14] P. Mell, T. Grance, The NIST definition of cloud computing (draft), NIST Spec. Publ. 800 (2011) 7.
[15] Y. Xing, Y. Zhan, Virtualization and cloud computing, in: Future Wireless Networks and Information Systems, Springer, 2012, pp. 305–312.
[16] B. Furht, Cloud computing fundamentals, in: B. Furht, A. Escalante (Eds.), Handbook of Cloud Computing, Springer, US, 2010, pp. 3–19. [online] Available at: http://dx.doi.org/10.1007/978-1-4419-6524-0_1.
[17] Xen, Available at: http://www.xenproject.org [August 21, 2013].
[18] VMWare, Available at: http://www.vmware.com [August 21, 2013].
[19] KVM, Available at: http://www.linux-kvm.org [August 21, 2013].
[20] The OpenMP API specification for parallel programming. Available at: August 12, 2013.
[21] The Message Passing Interface (MPI) standard. Available at: http://www.mcs.anl.gov/mpi [August 12, 2013].
[22] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, X. Qin, Improving mapreduce performance through data placement in heterogeneous hadoop clusters, in: 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, IPDPSW, 2010, pp. 1–9.
[23] S. Khalil, S.A. Salem, S. Nassar, E.M. Saad, Mapreduce performance in heterogeneous environments: A review, Int. J. Sci. Eng. Res. (IJSER) 4 (4) (2013) 410–416.
[24] X. Bu, J. Rao, C.-Z. Xu, Interference and locality-aware task scheduling for MapReduce applications in virtual clusters, in: Presented at the The ACM International Symposium on High-Performance Parallel and Distributed Computing, HPDC, New York, USA, 2013.
[25] Y. Fan, W. Wu, H. Cao, H. Zhu, X. Zhao, W. Wei, A heterogeneity-aware data distribution and rebalance method in Hadoop cluster, in: ChinaGrid Annual Conference (ChinaGrid), 2012 Seventh, 2012, pp. 176–181.
[26] J. Park, D. Lee, B. Kim, J. Huh, S. Maeng, Locality-aware dynamic VM reconfiguration on mapreduce clouds, in: Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing, 2012, pp. 27–36.
[27] W. Wang, K. Zhu, L. Ying, J. Tan, L. Zhang, A throughput optimal algorithm for map task scheduling in mapreduce with data locality, ACM SIGMETRICS Perform. Eval. Rev. 40 (2013) 33–42.
[28] M. Hammoud, M.F. Sakr, Locality-aware reduce task scheduling for mapreduce, in: 2011 IEEE Third International Conference on Cloud Computing Technology and Science, CloudCom, 2011, pp. 570–576.
[29] M. Li, D. Subhraveti, A. Butt, A. Khasymski, P. Sarkar, CAM: a topology aware minimum cost flow based resource manager for MapReduce applications in the cloud, in: Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing, Delft, The Netherlands, June 18–22, 2012.
[30] B. Palanisamy, A. Singh, L. Liu, B. Jain, Purlieus: Locality-aware resource allocation for MapReduce in a cloud, in: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC, 2011.
[31] M. Zaharia, A. Konwinski, A.D. Joseph, R.H. Katz, I. Stoica, Improving MapReduce performance in heterogeneous environments, in: OSDI, 2008, p. 7.
[32] X. Zhao, X. Dong, H. Cao, Y. Fan, H. Zhu, A parameter dynamic-tuning scheduling algorithm based on history in heterogeneous environments, in: ChinaGrid Annual Conference, ChinaGrid, 2012 Seventh, 2012, pp. 49–56.
[33] X. Sun, C. He, Y. Lu, ESAMR: An enhanced self-adaptive MapReduce scheduling algorithm, in: 2012 IEEE 18th International Conference on Parallel and Distributed Systems, ICPADS, 2012, pp. 148–155.
[34] Q. Chen, M. Guo, Q. Deng, L. Zheng, S. Guo, Y. Shen, HAT: history-based auto-tuning MapReduce in heterogeneous environments, J. Supercomput. (2013) 1–17.

Ching-Hsien Hsu received B.S. and Ph.D. degrees in Computer Science from Tung Hai University and Feng Chia University, Taiwan, in 1995 and 1999, respectively. From 2001 to 2002, he had been an assistant professor in the department of Electrical Engineering at Nan Kai College. He joined the department of Computer Science and Information Engineering, Chung Hua University in 2002, and has become an associate professor since August 2005. He has published more than 100 academic papers in journals, books and conference proceedings. His research interests include parallel and distributed processing, concurrent programming, parallelizing compilers, grid and pervasive computing. He is a senior member of the IEEE computer society.

Kenn D. Slagter received an NZCE in Electronics and Computer Technology from the Eastern Institute of Technology in 1996, a B.S. degree in Computer Science from the University of Waikato in 2000 and a Master of Computer Studies from the University of New England in 2007. In 2008 he joined the Department of Computer Science at National Tsing Hua University as a Ph.D. candidate. He has also over 8 years work experience in the private sector as a software engineer. His research interests include high performance computing, cloud computing and parallel and distributed systems. He is a student member of the IEEE computer society.

Yeh-Ching Chung received a B.S. degree in Information Engineering from Chung Yuan Christian University in 1983, and the M.S. and Ph.D. degrees in Computer and Information Science from Syracuse University in 1988 and 1992, respectively. He joined the Department of Information Engineering at Feng Chia University as an associate professor in 1992 and became a full professor in 1999. From 1998 to 2001, he was the chairman of the department. In 2002, he joined the Department of Computer Science at National Tsing Hua University as a full professor. His research interests include parallel and distributed processing, cloud computing, and embedded systems. He is a senior member of the IEEE computer society.