# Dynamic Data Partitioning and Virtual Machine Mapping: Efficient Data Intensive Computation

Kenn Slagter
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan
kennslagter@sslab.cs.nthu.edu.tw

Ching-Hsien Hsu
Department of Computer Science
Chung Hua University
Hsinchu, Taiwan
chh@chu.edu.tw

Yeh-Ching Chung
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan
ychung@cs.nthu.edu.tw

***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. This problem is exacerbated by heterogeneous environments.**

**In this paper we propose a method to improve MapReduce execution in heterogeneous environments. This is done by dynamically partitioning data during the Map phase and by using virtual machine mapping in the Reduce phase in order to maximize resource utilization.**

***Keywords—BigData; MapReduce; Hadoop; Virtual Machine; Heterogeneous environment; Cloud Computing; Parallel Computing.***

## I. INTRODUCTION

MapReduce is a programming model for creating distributed applications that can process big data using a large number of commodity computers. Originally developed by Google[1,2], MapReduce enjoys wide use by both industry and academia[3] via Hadoop[4]. 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[3,5]. However, the efficacy of MapReduce can be undermined by its implementation. For instance, Hadoop the most popular open source MapReduce framework[5] assumes all the nodes in the network to be homogenous. Consequently, Hadoop's performance is not optimal in a heterogeneous environment[6].

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. There are many reasons why MapReduce might execute in a heterogeneous environment. For instance, advances in technology might mean new machines in the network are different to old ones. Alternatively, MapReduce may be deployed on a hybrid cloud environment, where computing resources tend to be heterogeneous[7]. In summary, this paper presents the following contributions

- A method to improve mapper performance in a heterogeneous environment by dynamically partitioning 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 MapReduce. 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. Finally, in section 5, we present our conclusion and prospects for future work.

## II. MAPREDUCE

The purpose of MapReduce is to process large amounts of data on clusters of computers. At the heart of MapReduce resides two distinct programming functions, a map function and a reduce function[8]. It is the responsibility of the programmer to provide these functions. Two tasks known as the mapper and reducer 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*.

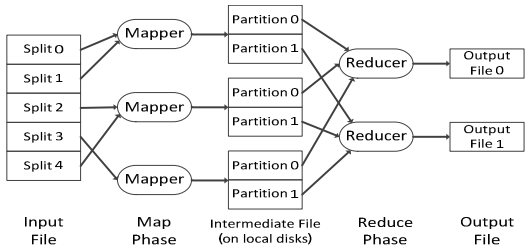


Figure 1. MapReduce data flow

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. During this process, data is stored locally in temporary intermediate file, as shown in Fig 1.

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

978-0-7695-5095-4/13 $31.00 © 2013 IEEE
DOI 10.1109/CloudCom.2013.134

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 in whatever way the user wants once all reducers have completed their workload.

## III. PROPOSED TECHNIQUES AND IMPLEMENTATION

The research model for this study is presented in Fig. 2, 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. The virtual machine may run a map task or a reduce task. Due to the heterogenerous nature of the environment, the processing capabilities of any particular virtual machine may differ from other virtual machines in the environment.

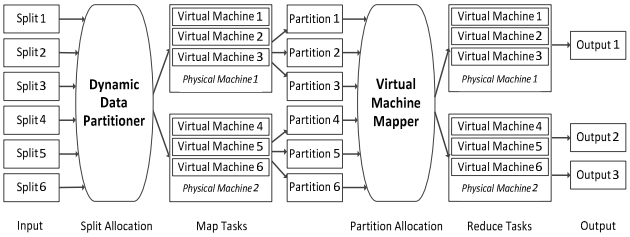


Figure 2. Dataflow of MapReduce model with the proposed dynamic data partitioner and a virtual machine mapper. Six Map tasks and three Reduce tasks run on virtual machines with differing processing capabilities.

### A. Dynamic Data Partitioning

In Hadoop, a MapReduce job begins by first reading a large input file. This file is usually stored on the Hadoop Distributed File System (HDFS). Since Hadoop assumes the environment is homogenous, the data from this file is split into fixed sized pieces. Hadoop then creates a mapper for each split. In a homogenous cluster each node has the same processing power and capabilities. In this case, each mapper will finish processing its split at approximately the same time. In a heterogeneous network, nodes that process faster than others will complete their work earlier.

Since data access rates between nodes on the HDFS are inconsistent due to issues of data locality, we propose a dynamic data partitioner that partitions data on a node irrespective of other nodes on the network. An example of the dynamic data partitioner is shown in Fig. 3. In this example, a 600GB file is used as input data. In this scenario, the data is divided up into six equal sized pieces, and sent to six virtual machines. Each of these virtual machines then executes a map task. Each virtual machine is given a value n that indicates the relative processing ability of that virtual machines VPU. This is based on the number of virtual processing units (VPU) of that virtual machine, and the physical machine it is running on. 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.

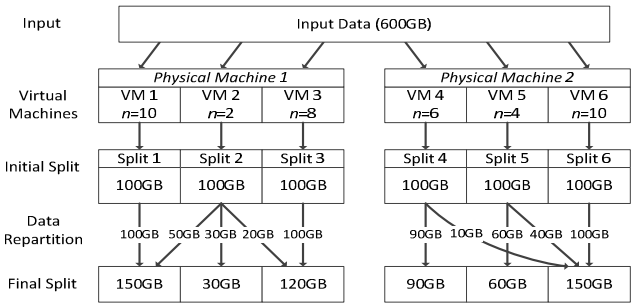


Figure 3. Dynamic Data Partitioner

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.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.splitSize = VM.processingSpeed * fragmentSize
11.   end for
12. end for
```

Once the initial input splits are designated to each virtual machine the DDP repartitions the data on each physical machine. On PM1 there is three virtual machines VM1, VM2 and VM3. VM1 has a VM processing rate of 10, VM2 has a VM processing rate of 2 and VM3 has a processing rate of 8. Each virtual machine has an initial split size of 100GB. Consequently, the total data size of the three virtual machines is 300GB, and the total 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:

$$\text{fragmentSize} = \text{totalDataSize} / \text{totalSpeed} \quad (1)$$

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.

$$\text{splitSize} = \text{VM.processingSpeed} * \text{fragmentSize} \quad (2)$$

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

TABLE I. SUMMARY OF INPUTSPLIT DISTRIBUTION

| Physical Machine | Virtual Machine | VPU | Initial Input Split | Final Input Split |
|---|---|---|---|---|
| 1 | 1 | 10 | 100GB | 150GB |
| | 2 | 2 | 100GB | 30GB |
| | 3 | 8 | 100GB | 120GB |
| 2 | 4 | 6 | 100GB | 90GB |
| | 5 | 4 | 100GB | 60GB |
| | 6 | 10 | 100GB | 150GB |

## B. *Virtual Machine Mapping*

In Hadoop, 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 will 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. 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.

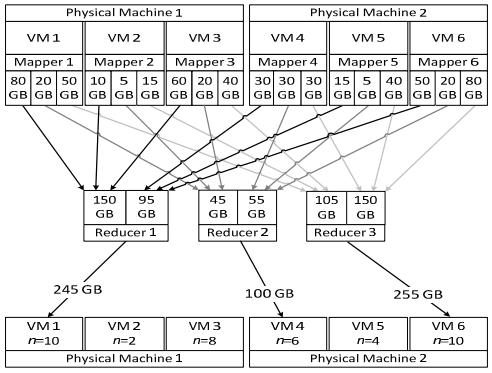


Figure 4. Virtual Machine Mapper

Fig. 4 shows a simple 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 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 each mapper on PM
6.      for i = 1 to NumberOfReducers
7.      for j = 1 to 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. //assign reducers to physical machines based on
    reducers[i].partition[j].size and available VM's
14. sortByBestfitAndPriority( reducers )
15. //assign reducer to fastest VM
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.   for i = 1 to PM.numberOfReducers
22.         SVM[i] = reducers[i]
23.   end for
24. end for
```

In Fig. 4 there are two physical machines. Both physical machines have three virtual machines each running a mapper. Each mapper produces three data partitions, which are 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. Since reducer 3 has more data stored on physical machine 2, it is assigned to the fastest virtual machine (VM6). Consequently, reducer 2 is assigned to the second fastest virtual machine (VM4). In this example, all of the reducers are allocated to a virtual machine on an appropriate physical machine. If there are no virtual machines available, the reducer is allocated to another physical machine. This is first done using the best fit selection method. If a physical machine has more reducers requesting a virtual machine than there are available virtual machines, the VMM has to locate the reducer to another physical machine. Instead of rejecting reducers based on a first come first served (FCFS), the VMM assesses the size of the data contributed to the reducer by each physical machine. The VMM then gives priority to those reducers with the greatest difference between their largest data contribution and their second largest data contribution.

## IV. EVALUATION

To evaluate the performance of the proposed technique, we implemented the dynamic data partitioner and virtual machine mapper and tested these methodologies with a 900GB randomly generated input data file on a simulated MapReduce environment. 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.

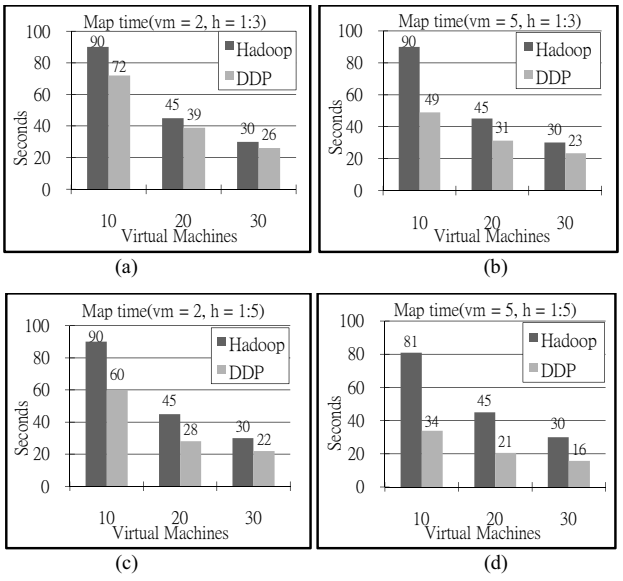


Figure 5. Map Completion Time: (a) low heterogeneity, vm = 2 (b) low heterogeneity, vm = 5 (c) high heterogeneity, vm = 2 (d) high heterogeneity, vm = 5

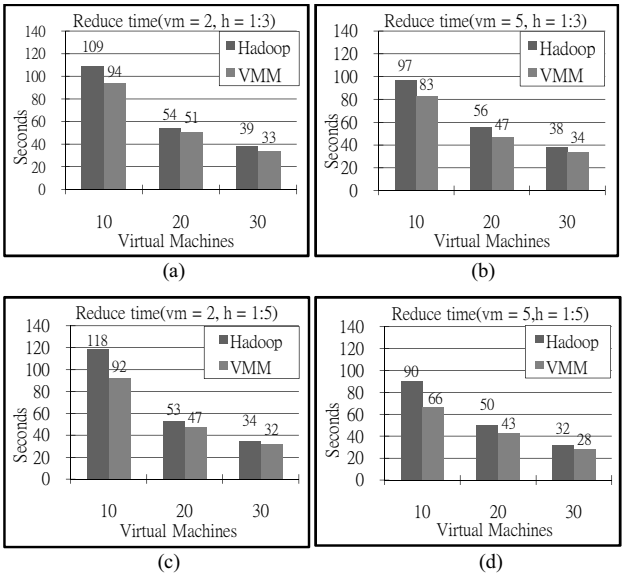


Figure 6. Reduce Completion Time: (a) low heterogeneity, vm = 2 (b) low heterogeneity, vm = 5 (c) high heterogeneity, vm = 2 (d) high heterogeneity, vm = 5

In Fig. 5, 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. 6, we explored reduce completion time. In the Reduce phase, the Virtual Machine Mapper selects which virtual machine a reducer should reside. Fig. 6(a) shows an improvement in reduce completion time of 14%, Fig. 6(b) shows an improvement in reduce completion time of 16%, Fig. 6(c) shows an improvement in reduce completion time of 23%, and Fig. 6(d) shows an improvement in reduce completion time of 29%. These simulation results show that our proposed methods save more time as heterogeneity increases.

## V. CONCLUSION AND FUTURE WORK

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. Simulation and experimental results show an improvement in MapReduce performance, improving map task completion time by up to 44% and reduce task completion time by up to 29%.

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

## REFERENCES


[1] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM,* vol. 51, 2008, pp. 107-113.

[2] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *ACM SIGOPS Operating Systems Review*, 2003, pp. 29-43.

[3] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon, "Parallel data processing with MapReduce: a survey," *ACM SIGMOD Record,* vol. 40, 2012, pp. 11-20.

[4] http://hadoop.apache.org

[5] J. Dittrich and J.-A. Quiané-Ruiz, "Efficient big data processing in Hadoop MapReduce," *Proceedings of the VLDB Endowment,* vol. 5, 2012, pp. 2014-2015.

[6] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and X. Qin, "Improving mapreduce performance through data placement in heterogeneous hadoop clusters," in Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on, 2010, pp. 1-9.

[7] S. Khalil, S. A. Salem, S. Nassar, and E. M. Saad, "Mapreduce performance in heterogeneous environments: A Review." *International Journal of Scientific and Engineering Research* (IJSER), vol. 4, issue 4, 2013, pp. 410-416.

[8] T. White, *Hadoop: The definitive guide*: 3rd Edition, O'Reilly Media, Inc., 2012.