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 bad situation by heterogeneous environments.

In this paper a methodology is properly to improve MapReduce execution in heterogeneous environments. It is carried out using 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;

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 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.

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.

![MapReduce data flow](image)

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.

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.

**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 heterogeneous nature of the environment, the processing capabilities of any particular virtual machine may differ from other virtual machines in the environment.

Since data access rates between nodes on the HDFS are inconsistent due to issues of data locality, we propose a dynamic data partitioner. 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.

![Figure 3. 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.](image)

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 machine 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 4. Example of MapReduce program execution with virtual machine VPU](image)
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 calculated. The machine the DDP repartitions the data on each physical machine. On PM1 there are 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:

![Image](image-url)

Table I - MAP VS. MATCH VS. PREGEL BASED

<table>
<thead>
<tr>
<th>TYPE</th>
<th>FILE</th>
<th>TIME [M]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ca.CondMat.txt</td>
<td>Match/Reduce</td>
<td>0.23</td>
</tr>
<tr>
<td>ca.CondMat.txt</td>
<td>Pregel</td>
<td>0.25</td>
</tr>
<tr>
<td>ca.CondMat.txt</td>
<td>Match/Reduce</td>
<td>0.23</td>
</tr>
<tr>
<td>com.youtube.ungraph.txt</td>
<td>Pregel</td>
<td>9.98</td>
</tr>
<tr>
<td>com.youtube.ungraph.txt</td>
<td>Match/Reduce</td>
<td>16.78</td>
</tr>
<tr>
<td>com.youtube.ungraph.txt</td>
<td>Map/Reduce</td>
<td>34.40</td>
</tr>
<tr>
<td>web.BerkStan.txt</td>
<td>Match/Reduce</td>
<td>24.70</td>
</tr>
<tr>
<td>web.BerkStan.txt</td>
<td>Map/Reduce</td>
<td>298.35</td>
</tr>
</tbody>
</table>

A closer analysis of Figure 3 reveals that data sets with less than a hundred thousand nodes play in the hands of Map/Reduce. 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.

![Image](image-url)

**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.

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 which eases recovery, but hampers performance [5]. However, with the relative youth of its successors, it is cur-rently the only framework that can actually recover from faults. Both distributed GraphLab and Stratosphere detect lost nodes quickly, however cannot recover...
from errors. Both display an error code on the console.

Only Stratosphere has a built-in recovery mechanism, which is at the time of writing unable to handle iterations properly. In Fig. 3 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.

Both scenarios using the analyzed vertex centric frameworks. The results for the Map/Reduce frameworks did not match the speed of their counterparts and have been left out. The first thing to notice is that GraphLab apparently comes from a multicore background and is still optimized for that. In that scenario, Hama performs worse than GraphLab by at least 20% for every tested data set. The gain shows up especially using large data sets in the multicore scenario, which is an important benefit.

In the distributed scenario, the gain is negligible for a huge data set like Wiki-talk, and even reversed for the Berkeley data. As previously mentioned, dense data set seems to exploit a shortcoming in the implementation of distributed GraphLab. For a complete analysis of GraphLab on a multicore machine see 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 both scenarios using the analyzed vertex centric frameworks. The results for the Map/Reduce frameworks did not match the speed of their counterparts and have been left out. The first thing to notice is that GraphLab apparently comes from a multicore background and is still optimized for that. In that scenario, Hama performs worse than GraphLab by at least 20% for every tested data set. The gain shows up especially using large data sets in the multicore scenario, which is an important benefit.

CONCLUSIONS

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.

REFERENCES