**MapReduce in the Cloud: Data-Location-Aware VM Scheduling**

Tung Nguyen and Weisong Shi

(Department of Computer Science, Wayne State University, Detroit, USA)

**Abstract**: We have witnessed the fast-growing deployment of Hadoop, an open-source implementation of the MapReduce programming model, for purpose of data-intensive computing in the cloud. However, Hadoop was not originally designed to run transient jobs in which users need to move data back and forth between storage and computing facilities. As a result, Hadoop is inefficient and wastes resources when operating in the cloud. This paper discusses the inefficiency of MapReduce in the cloud. We study the causes of this inefficiency and propose a solution. Inefficiency mainly occurs during data movement. Transferring large data to computing nodes is very time-consuming and also violates the rationale of Hadoop, which is to move computation to the data. To address this issue, we developed a distributed cache system and virtual machine scheduler. We show that our prototype can improve performance significantly when running different applications.

# 1 Introduction

Recently, the volume of data being generated as a result of simulation and data mining in the physical and life sciences has increased significantly. This trend has necessitated an efficient model of computation. Google MapReduce [1] is a popular computation model because it is suitable for data-intensive applications, such as web access logging, inverted index construction, document clustering, machine learning, and statistical machine translation. There are several implementations of MapReduce, including Phoenix [2], Sector/Sphere [3], open-source Hadoop [4], and Mars [5]. In fact, Hadoop is so popular that Amazon offers a separate service, called Elastic MapReduce (EMR), based on it. In the past few years, we have witnessed the fast-growing deployment of Hadoop for data-intensive computing in the cloud.

In our analysis in section 2, we found that Hadoop is not as efficient as expected when running in the cloud. The first drawback is virtual machine (VM) overhead, which includes JVM overhead because Hadoop was developed with Java. A Hadoop MapReduce job is typically executed on top of a JVM operated inside another VM if run in the cloud. Our experiment shows that the execution time of an application run on VMs is about four times longer than the execution time of the same application run on physical machines. The second drawback is the extra overhead created by data movement between storage and computing facilities in the cloud. In Elastic MapReduce, data has to be transiently moved online from Amazon's simple storage services (i.e. S3) to the Hadoop VM cluster. Hadoop is often used to process extremely large volumes of data, and transient movement of this data puts a great burden on infrastructure. Resources such as network bandwidth, energy, and disk I/Os can be greatly wasted. For example, when sorting 1 GB of data on our testbed, the time taken to move the data was 4.8 times longer than the time taken to sort it. In this work, we focus on the data-movement problem and do not deal with VM overhead reduction.

We designed and implemented a distributed cache system and VM scheduler to reduce this costly data movement. In a warm-cache scenario, which is usually occurs after the cloud has been running for a while, our system improved performance by up to 56.4% in two MapReduce-based applications in the life sciences. It also improved performance by 75.1% in the traditional Sort application and by 83.7% in the Grep application.

The rest of the paper is organized as follows: In section 2, we first show the inefficiency of EMR in terms of performance and data access. In sections 3 and 4, we describe the design and implementation of our distributed cache system for improving EMR data movement. The performance of our prototype is evaluated in section 5, and related work is discussed in section 6. Conclusions are drawn in section 7.

# 

# 2 Problem Statement

In this section, we explore how the current cloud and EMR system works and identify potential issues. In our case, the first question is: How do Cloud providers generally store user data? The next question we ask is: How does EMR work with respect to data processing?

## 2.1 Background

In this paper, we focus on an Infrastructure as a Service (IaaS) system, such as Amazon Cloud or Eucalyptus [6]. Because insight into Amazon Cloud is very limited, we have to use Eucalyptus, an open-source cloud computing platform that has a similar interface to Amazon EC2, a computing service, and Amazon S3, a storage service. Eucalyptus supports both ATA over Ethernet (AoE) and SCSI over Infiniband (iSCSI) storage networking standards, but the Amazon S3 architecture has not been published. Whenever possible, we run our experiments on Amazon.

We believe that S3 is in a storage-area network that is separate from EC2 for two reasons. First, the bandwidth between S3 and EC2 instances is smaller than that between EC2 instances [7]. Second, separation between computation and storage in data centers is a common design feature. S3 is designed specifically for storing persistent data and has strict requirements in terms of security, availability, reliability, and scalability. EC2 instances are usually used to store transient data which, if not moved to Amazon Elastic Block Store (EBS) or S3, is destroyed when the instance is terminated. In the Eucalyptus Community Cloud, the volumes and bucket directories (similar to EBS and S3) are also located in the front-end node, which is separate from the nodes that host VMs [8]. Further information about the differences between S3 and EBS can be found in [9].

To use EMR, the user first prepares the execution jar files (in the MapReduce framework) as well as the input data. They then create and launch a job flow and obtain the results. The job flow contains all the information, such as the number of instances, instance types, application jar and parameters, needed to execute a job.

## 2.2 Problem Identification

There are two issues with EMR: 1) performance degradation from using VMs and 2) overhead created by data movement. In this work, we focus on the data-movement problem and do not deal with VM overhead reduction.

Overhead created by the use of VMs instead of physical machines is a conventional problem that has been well studied [10]–[12]. Most solutions to this problem are based on general platforms, such as Xen, KVM and VMware, providing better virtualization. In [13]–[15], the limited performance of Hadoop on VMs was investigated. However, this investigation was problematic because the physical nodes had two quad-core 2.33 GHz Xeon processors and 8 GB of memory, and the VMs only had 1 VCPU and 1 GB of memory.

The second issue we deal with in this work is the overhead created by data movement. The above EMR workflow implies that the user data has to be transferred between S3 and EC2 every time a user runs a job flow. Although this transfer is free of charge from the user’s perspective, it consumes resources, such as energy and network bandwidth. In fact, the cost may be considerable because the MapReduce framework is often used for data-intensive computing, and the data to be processed is massive in scale. The cost of moving all this data is not negligible.

|  |  |
| --- | --- |
| a) CloudAligner |  |

b) Sort

Figure 1.Data movement vs. job execution.

We carried out two experiments to test our hypothesis that data movement is not a negligible part of job flow. Both experiments were done on our private cloud with Eucalyptus. The correlation between our private cloud and Amazon will be discussed in the next section. CloudAligner and Sort applications were executed with varied workloads in order to measure the execution and data-movement times. Sort is a benchmark built into Hadoop. **Fig. 1** shows that data movement is the dominant part of the Sort application (it is 4.8 times the execution time) and also an increasing part of the CloudAligner application. The largest amount of input data in the CloudAligner experiment, although extracted from the real data, is less than one-tenth the size of the real data. Also, the selected reference chromosome, chr22, is the smallest of other chromosomes. The data movement part in the CloudAligner experiment indicates that data movement is would also be significant if real-sized data was used. Therefore, it can be concluded that data-movement time tends to dominate execution time in data-intensive applications.

In general, the current EMR approach violates the rationale behind the success of Hadoop, i.e. moving computation to the data. Although optimized to perform on data stored on S3, EMR still performs better on HDFS (**Fig. 2**). **Fig. 2** shows the results of running CloudBurst on an EMR cluster of 11 small EC2 instances with different data sizes and data stored on HDFS and S3. CloudBurst is a MapReduce application used to map DNA sequences to the reference genome [16]. Like CloudAligner, this type of application is fundamental in bioinformatics. It is used because it is also an exemplar application of EMR.

Figure 2. Execution time of CloudBurst on HDFS and S3.

Data movement is time-consuming, and in EMR, data has to be moved every time a user executes a job flow. With the elasticity and transiency of the cloud, data and VMs are deleted after the job flow has finished. The situation is even worse when, after the first run, the user wants to tune the parameters and re-run the application on the same data set. The whole process of moving the huge data set would be triggered again. One may argue that the original job could be kept alive, and the EMR command line tool could be used to add steps with modified parameters. However, for simplicity many users only use the web-based tool, which is currently rather simple and does not support such features. Keeping the system alive also means the user keeps paying.

This wastes a lot of bandwidth. To overcome this, user persistent data and computation instances have to be close to each other. We propose a system that caches user data persistently in the physical hosts' storage (where the VMs of that user are hosted). This way, when the same user comes back to the system, the data is ready to be processed.

# 

# 3 System Design

As stated in the previous section, our goal is to improve the movement of data by EMR by reducing the amount of data to be transferred to the computing system from the persistent storage. To achieve this goal, the loaded data should be kept at the computing instances persistently so that it can be used later. When a user returns to the system, their VMs should be scheduled close to the data.

# 3.1 Terminology, Assumptions and Requirements

Before getting into the details of our solution, we first clearly identify terminologies, the target system, the intended applications, and requirements.

We use the terms VM, instance, node, and computing node interchangeably. The terms physical machine and physical hosts are likewise used interchangeably. The back-end storage is referred to as persistent storage services, such as S3 or Walrus. The front-end, or ephemeral storage, is the storage at the computing nodes. However, the front-end server is the head node of the Eucalyptus cloud.

The specific system to which we want to add our cache should be similar to EMR because EMR is a proprietary production system and therefore cannot be accessed.

The intended application of our system is the same as that of the MapReduce framework: a data-intensive application. Using our modified EMR system, a user would not see any change in the system except for improved performance. The interaction between the user and system is also unchanged. The user still needs to create a job flow and specify instances, parameters, and executable files.

The input assumption of our system, which is the same as that of EMR, is that the user data and its executions are already stored in the persistent storage (SAN), such as S3. Therefore, it is not necessary to deal with data availability, reliability, and durability here. If our cache does not contain enough data (replicas) of a user, it can always be retrieved from the back-end storage.

Like with Hadoop, the data we are targeting has the property of “write once, read many time” because it is often extremely large and difficult to modifying or editing. Usually, the files in such a system are read-only or append-only. Therefore, strong consistency between the cached data and back-end (S3) is not required.

Because the user data is stored in the physical machines that host the VMs of different users, the system needs to secure this data. In other words, the cache needs to be isolated from the local hard drives allocated to VMs.

# 3.2 Solution

VMs and all retrieved data vanish after the job flow has finished. Therefore, to reduce wasted data movement, this fetched data is cached persistently in the computing cluster. The data should not be deleted alongside the VM termination. VMs are hosted on physical machines, which reserve a part of their local hard drives for this cache. This part is separated from the partitions for VMs. Another option is to use EBS because it is also persistent.

Fig. 3 shows how a user views the system. Each user has a set of VMs, and each VM has its own cache. Many VMs and caches of different users can share the same physical host.

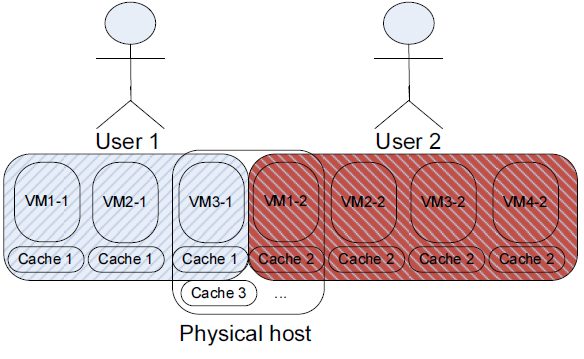


Figure 3. The system from user’s point of view.

However, the cluster comprises many physical servers, and most cloud users only have their VMs running on a very small part of the cluster. This means that a user’s data is only stored on a small number of physical servers in the cluster. There is no guarantee that a user’s VMs are hosted on the same set of physical servers between two different job flow executions. This may result in the cached data being unavailable to the user.

There are two possible solutions to this problem. We can move the missed retrieved data to the machines that host the user’s new VMs or we can modify the VM scheduler to host the user’s VMs on the same set of physical machines among different job flows. Each solution has its pros and cons. In the first solution, although the VM scheduler does not need to be modified, we need to keep track of the location of the cached data and the new VM hosts of all users. With the new VM-host mapping, we can then identify which parts of the cached data are missing and copy or move them to the new hosts. In the second solution, the data does not need to be moved, but the load in the system may become unbalanced because of the affinity of VMs for a set of physical servers. Even if the preferred physical machines of the return client do not have enough remaining resources, the system still has to put VMs into other available physical machines. In our implementation, we take the second approach. However, the delayed-scheduling approach [17] can also be taken to achieve both locality and fairness.

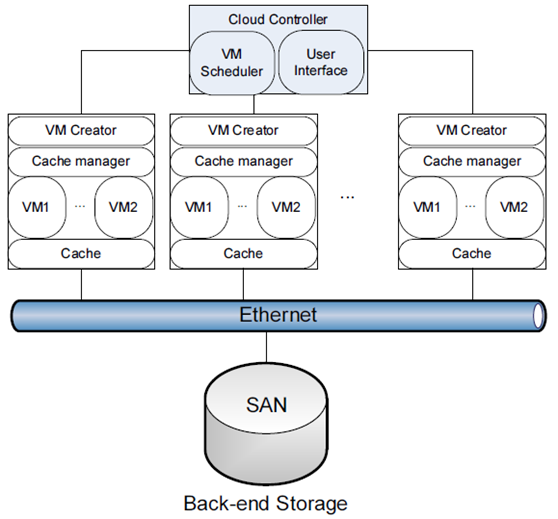


Figure 4. Logical view of the system.

Fig. 4 shows the logical view of the system. We employ the traditional master-slave model, which also matches with Eucalyptus (on which we implement our prototype). The master node, also called front-end in Eucalyptus, is the cloud controller in Fig. 4 and contains the web service interface and VM scheduler. The scheduler communicates with the nodes in order to start VMs on them. Each node also uses web services to handle requests from the scheduler. On the nodes are running VMs, the cache partition (system cache), the VM creator (Xen, KVM, libvirt), and the cache manager. The cache manager attaches the cache to the VM and also implements replacement algorithms, such as FIFO and LRU.

When a user submits a job, the VM scheduler system tries to allocate their VMs to the physical hosts that already contained their data. Otherwise, if the user is new and has not uploaded any data to the system (i.e. HDFS, not S3), the system can schedule their VMs to any available hosts that do not trigger the cache-replacement process (or only trigger the smallest part of it).

To ensure isolation, the cache is allocated to separate partitions on the physical hosts. During the cache-replacement process, the cache is automatically attached to the VMs according to the appropriate user, and it persists after the VM has been terminated. Another design-related decision is whether each user should have a separate partition in the hosts or if all users should share the same partition as a cache. For security and isolation reasons, we choose the former.

The size of the cache partition is important and can be varied or fixed. For simplicity, we fix the cache size for each user (user cache). However, the partition for the cache at a physical machine (system cache) can be dynamic. The same physical machine can host a different number of VMs depending on the VM types (m1.small, c1.large, etc.). The user cache size is also proportional to the VM types, although it is fixed. Different VM types have different fixed user cache sizes. To ensure fairness, all users have the same sized cache for the same type of VM.

The size of the system cache is also important. If it is too small, it does not efficiently reduce data movement because the system has to replace old data with new. The old data does not have many chances to be reused. If it is too large, part used for the ephemeral storage of the VMs (regular storage of the VMs) is reduced. Hence, the number of VMs available for hosting on a physical machine is also reduced. In fact, the cache size depends on the system usage parameters, such as average number of users and user cache size. If the system has too many simultaneous users or the user cache is too large, data replacement can be triggered too many times. We can extend the system cache or reduce the user cache to relieve this.

There are two cache-replacement levels in our system. The first is at the physical node and the second is across the whole system. When the local cache is full, the node should migrate the data in its cache system to other available machines. When the global cache is full, the old cache should be removed by following traditional policies, such as FIFO or LRU. There are two options available here: remove all the data of users or remove just enough data to accommodate the newest data.

The cache-replacement mechanism depends on cache availability and VM availability. These two factors are independent. A physical host may not be able to host new VMs, but it may still have available cache (and verse versa).

# 

# 4 Implementation

## 4.1 EMR versus Private Cloud

Because we do not have insight into EMR, we use an open-source private cloud to measure the steps in the job flow. Fig. 5 shows the execution time for our cloud and EC2 when running CloudBurst with the same data set and with the same VM configurations. We also used the same number of VMs with almost the same configurations. The correlation between our system and EMR is 0.85.

Figure 5. Execution time for EC2 and Eucalyptus.

**Fig. 5** shows that our system performs worse than EMR because of limitations in our network. According to [18] and [7], the bandwidth between EC2 instances (computing nodes) is 1 Gbps (100 MB/s), and the bandwidth between EC2 and S3 is 400 Mbit/s (50 MB/s). In our network, the average bandwidth between the physical nodes (computing nodes) is only 2.77 Mbit/s, between computing nodes and persistent storage node (Walrus) is about 2.67 Mbit/s, and between the client and front-end is 1.2 Mbit/s. CloudBurst itself is a network-intensive application [19].

# 4.2 Our Implementation

Generally, to implement the proposed system, the cache partition has to be created at the physical nodes; the VM creation script has to be changed to attach the cache to the VM; and the VM scheduler of the cloud has to be modified. Creating a partition in any operating system should be a minor task. Most cloud implementations have libvirt toolkit for virtualization because it is free, supports many different operating systems, and supports the main hypervisors, such as Xen, KVM/QEMU, VMWare, Virtual Box, and Hyper-V. Therefore, the VM cache attachment task can be applied to almost any cloud. Modification of the VM scheduler depends on the implementation of the cloud. However, many well-designed implementations enable the administrator to easily add new schedulers. Such implementations also allow configurability so that the desired scheduler can be selected. In our implementation, we use Hadoop 0.20.2 and Eucalyptus 2.0.3, and the EMI image is CentOS.

To add our cache system to Eucalyptus, we need to create a cache partition on each physical node and specify the fixed size for each type of instance. We then mount the cache system to a suitable point of the VM so that Hadoop can access the VM. In our prototype, the size of the user cache is only 1.2 GB. For simplicity, we also only ran experiments with one type of instance.

Although the user cache is fixed, the system cache is not. To enable dynamic sizing of the system cache partition, we use a logical volume manager (LVM). To attach the user cache as a block device in a VM, we modify the VM XML creator script of Eucalyptus. The block device (cache) is not actually mounted to the file system; therefore, it cannot be used yet.

To mimic EMR, we need to create a script receiving information like job flow. This script then invokes other scripts to create instances, start Hadoop, run the job, and terminate the instances. The standard Eucalyptus VM image (EMI) does not have Hadoop; therefore, we modify the EMI so that Hadoop can be installed and configured, and we also enable the EMI to run the user script. This enables us to automate the starting/stopping script easily and to mount the user cache to the directory used by HDFS.

So far, we have only mimicked the EMR and prepared the cache storage at the node. The main task in our system is to schedule user VMs close to their data. When a user comes back to the system, the data should already be in the Hadoop cluster. To realize this, we modified the Eucalyptus scheduler.

We added code to the Eucalyptus cluster controller to record the map between the user and their VM locations. Then, we modified the VM scheduler so that if it detects a returning user with the same number of instances requested (by looking at the recorded map), it schedules the user's VMs to the previous locations if possible. If there are no resources remaining for the new VMs, the system uses the default scheduling policy to move the corresponding cache to other available nodes and starts the VMs there. If there is no available cache slot in the whole system, we should swap the cache in the original node with the (oldest or randomly) one on the ``available to run VM'' node.

# 

# 5 Performance Evaluation

# 5.1 Experiment Setup

We implemented a prototype cache system on Eucalyptus. Our testbed consisted of 12 machines, the configurations of which are shown in **Table 1**.

**Table 1.** Testbed configuration

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Type** | **Number of Machines** | **CPU** | **Memory (GB)** | **HDD (GB)** | **OS** |
| Client | 1 | AMD 2 GHz | 6 | 250 | Ubuntu |
| Front-end | 1 | AMD Phenom II | 8 | 500 | CentOS |
| Computing | 10 | Xeon 2.80 GHz | 2 | 40 | CentOS |

The workloads used in this section derive from CloudAligner, CloudBurst, Hadoop Sort benchmarks and Grep, which is a built-in MapReduce application used to search for an expression on the input files. The input for Sort is generated by RandomWriter. The input for Grep comprises the Hadoop log files and a Shakespeare play. The search expression is 'the\*'.

**Table 2.** Size of the input read files

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input split (kB)** | **100** | **200** | **400** | **600** | **800** | **1000** | **2000** |
| Writable (MB) | 4.3 | 8.6 | 18 | 26 | 35 | 43 | 86 |
| Text (MB) | 5.2 | 11.0 | 21 | 32 | 42 | 53 | 106 |

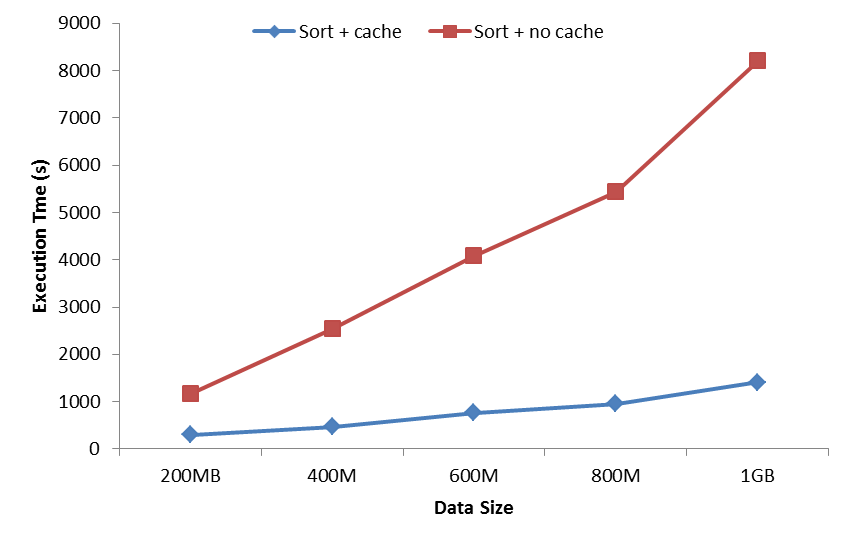
The data for both CloudBurst and CloudAligner comprises two pieces. The first piece is the read sequences produced by a sequencer such as Illumina GAII, HiSeq 2000 or Pacific Biosciences. We obtained real data from the 1000 Genomes project; in particular, we used the accession SRR035459 file, which is 956 MB. We extracted subsets from that file to use in our experiments. Table 2showsthe size and number of input splits as well as the size of the data in MapReduce Writable format and size of the original text file. Another piece of data is the reference genome. For this, we choose chromosome 22 of the human genome (with original text size of 50 MB, and 9.2 MB of this is in MapReduce Writable format). This data is small and is only used here as a proof of concept. The real data is much larger. For example, there are 22 chromosomes in the human genome and normally, the alignment software needs to align the read to all of these.

# 5.2 Improvements

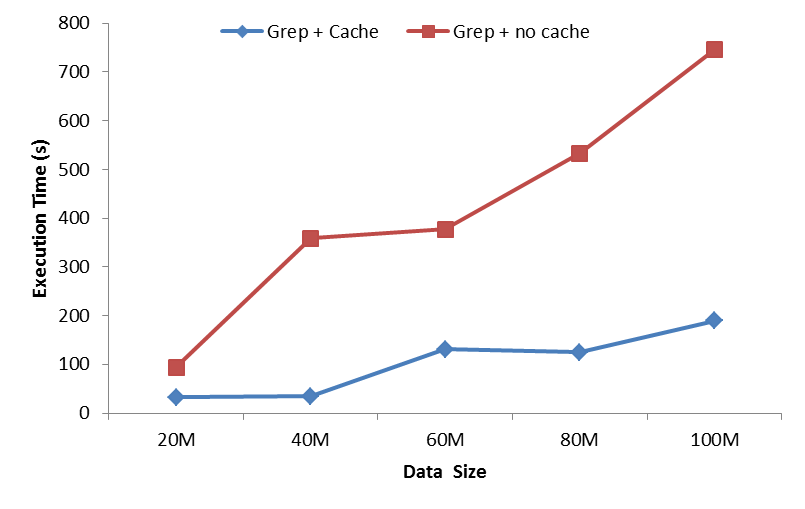
To show the improvement in performance brought about by our cache system and the VM scheduler, we should ideally run experiments on data stored in Walrus and compare the results with those obtained using HDFS because with a warm cache, the data already exists in HDFS. However, the jets3t 0.6 library in the Hadoop version we used does not support communication with Walrus. As a result, we took the following two approaches.

First, the data was moved from the Walrus server to HDFS, and data-movement durations were recorded. Without the cache, the data always has to be moved in; therefore, the execution time of a job flow should include this data-movement time. This approach is called addition. Fig. 6 shows the performance of CloudBurst and CloudAligner when using and not using cache. When the size of the data increases, especially after 800 k, the difference between using and not using cache is wider because the data-movement time increases faster than the processing time. Fig. 7 supports this observation. If the data is large and time to process it is small (due to MapReduce), the benefit of using our cache is greater. For example, in the Sort experiment, moving 1 GB of data to the system takes 6795 s in our network, but sorting it takes only 1405 s.

Figure 6. Performance of the CloudAligner and CloudBurst applications with and without warm cache when addition method is used.



1. Sort

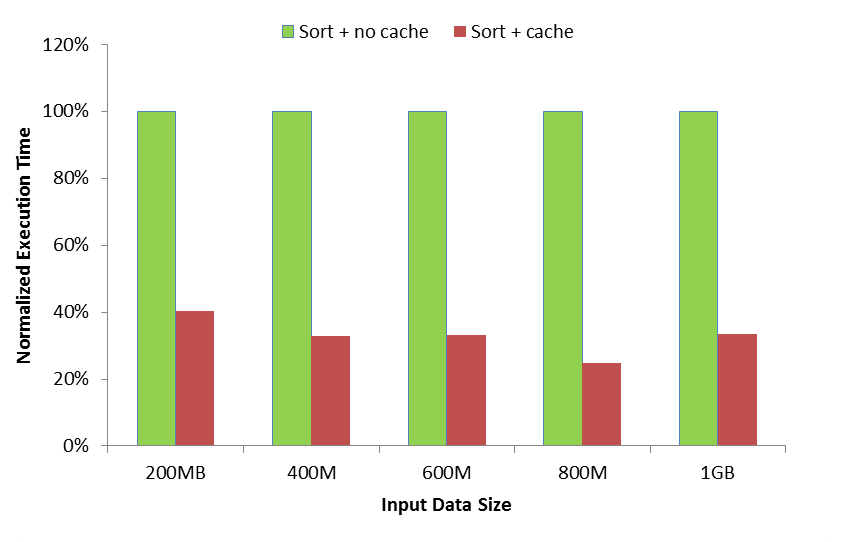


1. Grep

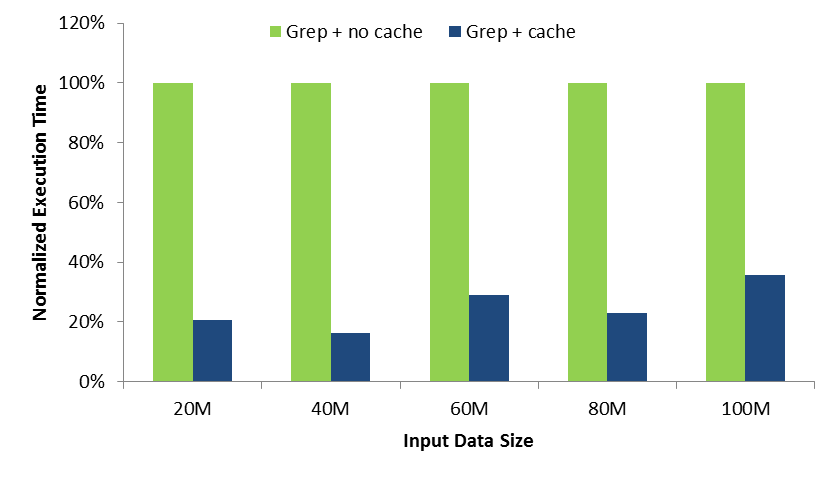
Figure 7. Performance of the Sort and Grep with and without warm cache when addition method is used.

Second, we compare the results of running the same application with data in HDFS and in Amazon S3. In this case, the computing nodes are in our testbed, not at Amazon. We use our local cluster to process the data from HDFS and S3. This method is called HDFS\_S3. The execution times for CloudBurst and CloudAligner with different configurations are shown in Fig. 8. Figs. 6 to 8 show the actual time to move and process the data; however, **Fig. 9** shows the relative performance for the applications without cache and with warm cache. This helps us see to what degree our system improves performance. For CloudAligner and CloudBurst, our system improves performance by up to 56.4% and 47.4%, respectively. Performance improvement in Sort and Grep is more significant because these need less time to process larger data. In particular, the Sort application can benefit up to 75.1%, and the number of the Grep application is 83.7%. This result makes sense because all Grep does is just to cut the large data into small parts and to search each part in parallel for the pattern linearly.

Figure 8. Performance of CloudAligner and CloudBurst with warm cache when HDFS\_S3 method is used.



1. Sort



1. Grep

Figure 9. Performance of Sort and Grep with warm cache when HDFS\_S3 method is used.

# 5.3 Overhead and Scalability

To the old system we introduced new images, added a cache partition to the VM, mounted the cache partition, started Hadoop, and modified the scheduler. We made modifications in three main areas: the cluster controller, the node controller, and the VM. In terms of VM startup time, overhead is negligible but may arise from the differences between the modified and original EMIs. These differences are in the installation of Hadoop, the VM startup xml (libvirt.xml), and the mounting script. This overhead is negligible because the installation only needs to be done once, and the size of the image does not change after this installation. The mounting script also contains only one Linux mount command, and the VM startup xml only has one additional device. In addition, the VM startup time is very small compared with the time needed to prepare for the instance (i.e. copy root, kernel, ramdisk files from the cache or from Walrus, create ephemeral file, etc).

In terms of execution time, overhead is also negligible because our additional scheduler is active only when the user has already visited the system. In addition, our scheduler improves the scheduling process because it does not need to spend time iterating each node in order to find an available node for the user request.

The complexity of our scheduler is given by , where *U* is the number of EMR users, *M* is the maximum number of physical nodes that host a user's VMs, and *N* is the total number of physical hosts of the cloud.

Scalability can be expressed in terms of the number of users and size of the system (i.e. number of physical nodes). In terms of system size, our solution does not affect the scalability of the existing cloud platform. If *U* is not taken into account, our scheduler has the same complexity as the original schedulers, which is given by .

However, because we store user VM schedule plans, our approach does not scale well in terms of the number of users. Our work focuses on MapReduce applications only, so our cache system only serves a proportion of cloud users (not all cloud users use EMR).

Our cache solution is a best-effort solution, not an optimal one. This means that data can still move between different physical nodes if the previously scheduled physical nodes are not available. Without our solution, such movement would occur always.

# 6 Related Works

There is a distributed cache built into Hadoop [20], but our cache is different. With the Hadoop distributed cache, files are already in the HDFS. In our cache, the files are in the back-end storage system (S3). The data in [20] is cached between the map tasks and reduce tasks. In addition, the target platform in [20] is the small cluster, but ours is the cloud. Another recent work that describes cache in the cloud is [21]. Although this work mainly focuses on improving the caching of the VM image template, its concept can be directly applied to our case in order to further improve overall system performance.

EBS is suitable to use as our cache because it is independent from the instances. However, the current EMR does not support it. To use EBS, a user has to manually create suitable AMI with Hadoop, start it, attach the EBS, configure, start Hadoop, and run a job. If the user wants to run their job again, the whole process has to be repeated manually, even though the data remains on the EBS volumes.

The performance of MapReduce has received much attention recently. For example, [22] and [23] argued that Hadoop was designed for a single user and described ways of improving the performance of MapReduce in multiuser environment. A more recent work on Hadoop MapReduce for data-intensive applications is [24], but the authors evaluate the performance of data-intensive operations, such as filtering, merging, and reordering. Also, their context is high-performance computing, not cloud computing.

The closest work to ours on MapReduce and the cloud is [25]. In this work, Hamoud et al. also exploit data locality to improve the performance of the system. The main difference between their work and ours is that they schedule reduce tasks whereas we schedule VMs. In their evaluations, the authors of [26] use MapReduce, the cloud, and bioinformatics applications and in this context, their work is similar to ours. The authors propose Azure MapReduce as a MapReduce framework on Microsoft Azure Cloud infrastructure. Interestingly, their work supports our observation that running MapReduce on a bare metal cluster results in better performance than running MapReduce on a cloud-based cluster. It did not try to bridge this gap like ours.

# 7 Conclusion and Future Work

We have highlighted several existing issues with MapReduce in the cloud. We have also proposed a distributed cache system and VM scheduler to address the data-movement issue. By doing this, we can improve system performance. We have implemented a prototype of the system with Eucalyptus and Hadoop, and the experimental results show significant improvement in performance for certain applications.

If the user application associates with an extremely large data set that cannot fit into our distributed cache, our solution can still help reducing scheduling overhead. In this case, moving data from back-end storage is inevitable.

In the future, we plan to make Hadoop work with Walrus, obtain a real trace to measure cache hit or miss ratios, implement cache movement across the whole system, and monitor the removed data.

**References**

[1] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM,* vol. 51, no. 1, pp. 107-113, 2008. doi: [10.1145/1327452.1327492](http://dx.doi.org/10.1145/1327452.1327492" \t "_self).

[2] R. M. Yoo, A. Romano and C. Kozyrakis, "Phoenix rebirth: Scalable MapReduce on a large-scale shared-memory system," *IEEE Workload Characterization Symposium*, pp. 198-207, 2009. doi: [10.1109/IISWC.2009.5306783](http://dx.doi.org/10.1109/IISWC.2009.5306783" \t "blank).

[3] Y. Gu and R. L. Grossman, "Sector and Sphere: the design and implementation of a high-performance data cloud," *Philosophical Transactions of the Royal Society A: Mathematical,Physical and Engineering Sciences,* vol. 367, no. 1897, pp. 2429-2445, 2009. doi: 10.1098/rsta.2009.0053.

[4] Apache Hadoop[Online].Available: http://wiki.apache.org/hadoop/

[5] B. He, W. Fang, Q. Luo, N. K. Govindaraju and T. Wang, "Mars: a MapReduce framework on graphics processors," in *Proc.of the 17th international conference on Parallel architectures and compilation techniques*, New York, NY, USA, 2008, pp. 260-269. doi: [10.1145/1454115.1454152](http://dx.doi.org/10.1145/1454115.1454152" \t "_self).

[6] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Proc. of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, Shanghai, May 18-21, 2009, PP. 124-131. doi: [10.1109/CCGRID.2009.93](http://dx.doi.org/10.1109/CCGRID.2009.93" \t "blank).

|  |  |
| --- | --- |
|  |  |

[7] T. Von Eicken, “ Amazon EC2 Network and S3 performance.”Oct, 2007.

[8] File system layout[Online]. Available: http://open.eucalyptus.com/learn/InstallingECC

[9] Differences between S3 and EBS[Online]. http://www.cloudiquity.com/2009/03/differences-between-s3-and-ebs/

[10] W. Huang, M. Koop and D. Panda, "Efficient one-copy MPI shared memory communication in virtual machines," in *Cluster Computing, 2008 IEEE International Conference on*, Tsukuba, Sept. 29-Oct.1 2008,pp.107-115. doi: [10.1109/CLUSTR.2008.4663761](http://dx.doi.org/10.1109/CLUSTR.2008.4663761" \t "blank).

[11] J. Liu, W. Huang, B. Abali and D. Panda, "High performance VMM-bypass I/O in virtual machines," in *Proc. of the USENIX Annual Technical Conference*, 2006.

[12] W. Huang, J. Liu, B. Abali and D. K. Panda, "A case for high performance computing with virtual machines," in *Proc. of the 20th annual international conference on Supercomputing*, New York, NY, USA, 2006, pp.125-134. doi: [10.1145/1183401.1183421](http://dx.doi.org/10.1145/1183401.1183421" \t "_self).

[13] S. Ibrahim, H. Jin, B. Cheng, H. Cao, S. Wu and L. Qi, "CLOUDLET: towards mapreduce implementation on virtual machines," in *Proc. of the 18th ACM international symposium on High performance distributed computing*, New York, NY, USA, 2009, pp.65-66. doi: [10.1145/1551609.1551624](http://dx.doi.org/10.1145/1551609.1551624" \t "_self).

[14] S. Ibrahim, H. Jin, L. Lu, L. Qi, S. Wu and X. Shi, "Evaluating MapReduce on Virtual Machines: The Hadoop Case," in *Proc. of the 1st International Conference on Cloud Computing*, Berlin, Heidelberg, 2009, pp. 519-528. doi: 10.1007/978-3-642-10665-1\_47.

[15] J. Shafer, S. Rixner and A. Cox, "The Hadoop distributed filesystem: Balancing portability and performance," in *Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on*, 2010, pp. 122-133. doi: [10.1109/ISPASS.2010.5452045](http://dx.doi.org/10.1109/ISPASS.2010.5452045" \t "blank).

[16] M. Schatz, "CloudBurst: highly sensitive read mapping with MapReduce," *Bioinformatics,* vol. 25, no. 11, pp. 1363-1369, 2009. doi: 10.1093/bioinformatics/btp236.

[17] M. Zaharia, D. Borthakur, J. Sarma, K. Elmeleegy, S. Shenker and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in *EuroSys 2010*, New York, 2010, pp. 265-278. doi: [10.1145/1755913.1755940](http://dx.doi.org/10.1145/1755913.1755940" \t "_self).

[18] T. Von Eicken, “ Amazon EC2 Network and S3 performance.”Oct, 2007.

[19] T. Nguyen, W. Shi and D. Ruden, "CloudAligner: A fast and full-featured MapReduce based tool for sequence mapping," *BMC Research Notes,* vol. 4, no. 1, pp. 171, 2011.

[20] S. Zhang, J. Han, Z. Liu, K. Wang and S. Feng, "Accelerating MapReduce with Distributed Memory Cache," in *Parallel and Distributed Systems (ICPADS), 2009 15th International Conference on*, Shenzhen, China, 2009, pp. 472-478. doi: [10.1109/ICPADS.2009.88](http://dx.doi.org/10.1109/ICPADS.2009.88" \t "blank).

[21] D. Jeswani, M. Gupta, P. De, A. Malani and U. Bellur, "Minimizing Latency in Serving Requests through Differential Template Caching in a Cloud," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, Honolulu, HI, 2012, pp. 269-276. doi: [10.1109/CLOUD.2012.17](http://dx.doi.org/10.1109/CLOUD.2012.17" \t "blank).

[22] S. Seo, I. Jang, K. Woo, I. Kim, J. Kim and S. Maeng, "Hpmr: Prefetching and pre-shuffling in shared mapreduce computation environment," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, New Orleans, LA, 2009, pp. 1-8. doi: [10.1109/CLUSTR.2009.5289171](http://dx.doi.org/10.1109/CLUSTR.2009.5289171" \t "blank).

[23] M. Zaharia, D. Borthakur, J. Sarma, K. Elmeleegy, S. Shenker and I. Stoica, "Job scheduling for multi-user mapreduce clusters," *EECS Department,* University of California, Berkeley, Tech. Rep. UCB/EECS-2009-55, Apr, 2009,pp. 2009-55.

[24] Z. Fadika, M. Govindaraju, R. Canon and L. Ramakrishnan, "Evaluating Hadoop for Data-Intensive Scientific Operations," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, Honolulu, HI, 2012, pp. 67-74. doi: [10.1109/CLOUD.2012.118](http://dx.doi.org/10.1109/CLOUD.2012.118" \t "blank).

[25] M. Hammoud, M. S. Rehman and M. F. Sakr, "Center-of-Gravity Reduce Task Scheduling to Lower MapReduce Network Traffic," in *IEEE CLOUD*, Honolulu, HI, 2012, pp. 29-58. doi: [10.1109/CLOUD.2012.92](http://dx.doi.org/10.1109/CLOUD.2012.92" \t "blank).

[26] T. Gunarathne, T.-L. Wu, J. Qiu and G. Fox, "MapReduce in the Clouds for Science," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, Indianapolis, IN, 2010, pp. 565-572. doi: [10.1109/CloudCom.2010.107](http://dx.doi.org/10.1109/CloudCom.2010.107" \t "blank).

**Biographies**

**Tung Nguyen** ([tnguyen@i-a-i.com](mailto:tnguyen@i-a-i.com)) is a research scientist at Intelligent Automation Inc. He plays a key role in many projects on data-intensive distributed processing, cloud computing, and massive data analysis using topological features. Dr. Nguyen received his PhD degree from Wayne State University in 2012. He received his BS and MS degrees in computer science and engineering from Ho Chi Minh City University of Technology, Vietnam, in 2001 and 2006. His research interests include green computing, cloud computing, data-intensive computing, and application of cloud computing to life sciences. He has published several papers on computer science and bioinformatics and has been published in the proceedings of *OSDI* and in *NPC, SUSCOM*, and *BMC Frontiers Genetics* journals. He has also been a peer reviewer at many conferences, including *Euro-Par* and *CollaborateCom*. His homepage is <http://www.cs.wayne.edu/tung/>

**Weisong Shi** ([weisong@wayne.edu](mailto:weisong@wayne.edu)) is a professor of computer science at Wayne State University.  He received his BS degree in computer engineering from Xidian University in 1995. He received his PhD degree in computer engineering from the Chinese Academy of Sciences in 2000. His research interests include computer systems, mobile computing, and cloud computing. Dr. Shi has published 120 peer-reviewed journal and conference papers and has an H-index of 24. He has been the program chair and technical program committee member of numerous international conferences, including WWW and ICDCS. In 2002, he received the NSF CAREER award for outstanding PhD dissertation (China). In 2009, he received the Career Development Chair Award of Wayne State University. He has also won “Best Paper Award” at *ICWE’04*, *IPDPS’05*, *HPCChina’12*, and *IISWC’12*.