Improving Hadoop Service Provisioning in A Geographically Distributed Cloud

Qi Zhang, Ling Liu, Kisung Lee, Yang Zhou
College of Computing, Georgia Tech
{qzhang90, ling.liu, kslee, yzhou86}@cc.gatech.edu

Aameek Singh, Nagapramod Mandagere, Sandeep Gopisetty,
Gabriel Alatorre
IBM Research – Almaden
{aameek.singh, pramod, sandeep.gopisetty, galatorr }@us.ibm.com

Abstract—*With more data generated and collected in a geographically distributed manner, combined by the increased computational requirements for large scale data-intensive analysis, we have witnessed the growing demand for geographically distributed Cloud datacenters and hybrid Cloud service provisioning, enabling organizations to support instantaneous demand of additional computational resources and to expand in-house resources to maintain peak service demands by utilizing cloud resources. A key challenge for running applications in such a geographically distributed computing environment is how to efficiently schedule and perform analysis over data that is geographically distributed across multiple datacenters. In this paper, we first compare multi-datacenter Hadoop deployment with single-datacenter Hadoop deployment to identify the performance issues inherent in a geographically distributed cloud. A generalization of the problem characterization in the context of geographically distributed cloud datacenters is also provided with discussions on general optimization strategies. Then we describe the design and implementation of a suite of system-level optimizations for improving performance of Hadoop service provisioning in a geo-distributed cloud, including prediction-based job localization, configurable HDFS data placement, and data pre-fetching. Our experimental evaluation shows that our prediction based localization has very low error ratio, smaller than 5%, and our optimization can improve the execution time of Reduce phase by 48.6%.*

Keywords: Geographically distributed cloud, Hybrid cloud, Cross-cloud Hadoop deployment, Performance optimizaiton

I. Introduction

A fundamental innovation of cloud computing is the transformation of the arrangement of computing resources from static, long term and high upfront ownership investment based systems to dynamic provisioning systems. By providing cloud infrastructure and platform as services dynamic resource sharing across organizational and geographical boundaries can be enabled seamlessly to respond to changing demands and requirements. As a result, we witness the growing attentions of hybrid cloud and multi-datacenter cloud deployment from both industry and academic research communities. These cloud technologies offer solutions to deal with high velocity and high volume of big data generated from geographically dispersed sources while providing real-time experience to a broader range of cloud consumers. More companies, such as IBM, Google, and Facebook, are managing multiple cloud-based datacenters, which are usually geographically dispersed, to deal with the increasing computational requirements on large-scale data intensive analysis while providing guaranteed low latency to their customers. Furthermore, recent study on efficient energy management for datacenters [1] shows that compared with putting all the infrastructures into a single datacenter, it is more energy efficient to divide a single high-density datacenter into multiple smaller datacenters, and keep low energy costs for each of them..

We argue that there are two categories of technical challenges for developing a geographically distributed cloud with high performance and high availability guarantee. The first category of challenges is centered on adapting and extending existing cloud computing software platform. Example key questions include (i) what additional capabilities should be provided on top of existing cluster computing infrastructure that is typically configured, allocated, managed and optimized in a single cloud datacenter; (ii) how should existing cluster computing facilities be extended to provide efficient execution runtime environment for applications running in a geographically distributed cloud. The second category of challenges is centered on new technologies for enabling fast interconnection between cloud datacenters. The example efforts include Facebook Prism [12], Google Megastore [2] and Spanner [6], HP FlexFabric Interconnect [14], Cisco VPLS [13], enabling virtualization of heterogeneous compute, storage, network resources through a common orchestration interface using technology, such as storage virtualization, network virtualization and software defined networking.

In this paper we focus on addressing the two research questions mentioned in the first category. Concretely, the cloud computing resource management today is tightly coupled with the autonomous administration of each physical datacenter. It discourages resources sharing and limits the utilization efficiency of datacenter resources. For example, when one of the datacenters is overloaded and unavailable, the inability of automated off-loading of some workloads to other datacenters can significantly hurt the competitiveness of both cloud service providers and cloud consumers. In addition, although it is feasible to deploy a multi-datacenter Hadoop cluster at the application layer and run Hadoop workloads across geographically distributed datacenters, the current Hadoop solutions fail to provide resource-aware job scheduling and task scheduling for multi-datacenter Hadoop cluster instances that are running in a geographically distributed cloud, comprising of multiple autonomous and possibly heterogeneous data centers. This limitation prevents seamless resource utilization across datacenters, and can lead to unpredictably poor performance for Hadoop MapReduce jobs. For example, current Hadoop MapReduce solutions do not differentiate cluster nodes at a local datacenter from cluster nodes at a remote datacenter in its job scheduler and its task scheduler, which leads to copying the same piece of data into multiple datacenters [2, 3, 4]. This can make the cost of maintaining data consistency prohibitively high [5]. At the same time, transferring large amount of data across datacenters can be unmanageably time consuming.

Bearing these problems in mind, in this paper we first conduct analytical and experimental study to identify the performance issues inherent in a geographically distributed cloud by comparing multi-datacenter Hadoop deployment with single-datacenter Hadoop deployment. We provide a generalization of the problem characterization in the context of geographically distributed cloud datacenters and discuss the general optimization strategies. Then we describe the design and implementation of a suite of system-level optimizations designed specifically for improving the performance of Hadoop service provisioning in a multi-datacenter cloud: (i) prediction-based MapReduce job localization, (ii) configurable HDFS data placement, and (iii) Map input pre-fetching. To the best of our knowledge, there is no comprehensive study of multi-datacenter Hadoop deployments from job and task scheduling perspective. The experimental evaluation shows that our prediction based localization has very low error ratio, less than 5%, and our optimization can improve the execution time of Reduce phase by 48.6%.

The rest of this paper is organized as follows. Section 2 reviews the basic concepts and notations and identifies the main performance bottlenecks through experimental analysis. Second 3 presents our solution approach for improving Hadoop service provisioning in a geographically distributed cloud. We evaluate the effectiveness of our solution approach in Section 4, outline the related work in Section 5 and conclude the paper in Section 6.

II. Design Overview

A. *Problem Definition*

We define a geographically distributed cloud as a virtual cloud datacenter that manages multiple geographically dispersed datacenters. Advantages of such a geo-distributed cloud include the capability of coordinated provisioning of computing resources from multiple autonomous datacenters to provide cloud consumers the configurable ability to processing and analyzing large scale datasets by moving computation to data. Furthermore, the system facility to configure and compute across datacenters seamlessly can further enhance the capability for hybrid cloud service provisioning. One of the challenges in developing such system facility is the fact that most of the jobs running across geographically dispersed datacenters may not always deliver the same level of performance guarantee as those running within a single datacenter for at least two reasons. First, the network I/O bandwidth between geographically disparate datacenters is typically less predictable and significantly slower than intra-datacenter bisection network bandwidth. Second, the larger and the more frequent the data transfer is between datacenters, the larger performance degradation the cross-datacenter job execution will have. Thus, our goal is to develop system-level facilities that enable efficient and seamless cross-datacenter big data processing in a geo-distributed cluster. In this paper, *a geo-distributed cluster* refers to a large cluster that is established by including multiple sub-clusters, each sub-cluster is from a different datacenter and has its own namespace. We use *sub-cluster* to refer to a cluster setup within a given datacenter. This architecture brings several advantages. First, more flexible services can be provided when sub-clusters are configured into a geo-distributed cluster, because resources can be shared among these sub-clusters to handle cloud burst [15] and load balancing. Second, it enables across-datacenter computing for content and data sharing and increased failure resilience. Our solution approach should meet the following three system properties. First, this geo-distributed cluster should be transparent to existing applications in terms of both usability and functionality. Second, failures in one sub-cluster should not affect the routine execution of the other clusters no matter they are located in the same datacenter as the failed cluster or not. Third, sub-clusters in each single datacenter should be able to provide the same functionality and performance as an independent cluster in the presence of geo-distributed cluster instances.

B. *HDFS and Hadoop MapReduce*

HDFS (Hadoop Distributed File System) is an open source distributed file system designed to run on a cluster of commodity hardware. HDFS stores each file as a sequence of blocks with the same size and each block is replicated for fault tolerance. HDFS supports distributed computing by a master-slave based coordination architecture, where the master node, called *NameNode*, stores the metadata and synchronize the slaves to perform distributed computation. And all the raw data are stored and processed on the slave nodes, called *DataNode*.

Hadoop MapReduce is an open-sourced software and distributed multi-phase programming framework that allows processing big data in parallel on large HDFS clusters. Each Hadoop MapReduce job runs in two stages: Map and Reduce. The Reduce stage consists of three phases: shuffle, sort, and reduce. There is a *JobTacker* runing on the master node, which is responsible for job scheduling and task scheduling. And a *TaskTracker* runs on each of the allocated slave nodes for launching and executing the Map tasks and Reduce tasks assigned by the *JobTracker*. We identify the following three components as the critical sections for improving the performance of a MapReduce job running in a geo-distributed cluster: *input splits of Map tasks, output records of Reduce tasks, and execution of shuffle phase.*

There are three categories of Map tasks according to where the input split of a Map task locates: The first category is the *node-local Map task*, which reads the input split stored locally at the same physical node as the Map task is running. This is the most efficient type of Map tasks, since their input can be retrieved from local disk. The second category is the *datacenter-local Map task*, which reads the input split not from the same compute node as the task is running on, but from the node that is located in the same datacenter. Therefore, this input split has to be transferred within the datacenter before the Map task starts. The third category is *datacenter-remote Map task*, which reads the input split located in a different datacenter. MapReduce framework schedules the *node-local Map task* as the highest priority, but the other two categories of Map tasks also exist when a node without any local input split is still asking for Map tasks.

The output records of each Reduce task will be written through HDFS to the stable storage, and each block needs to be replicated to three or more nodes within the HDFS cluster. We use the term *target nodes* to refer to the nodes that HDFS chooses to store the replicas of a certain block. Since a Reduce task returns successfully only when all of its output replicas are written correctly to the stable storage, the selection of target nodes can affect the performance of Reduce tasks in a geo-distributed cluster. For example, if one of the target nodes is located in a different datacenter, this Reduce task is expected to execute for a relatively longer time since one of its output records will be written across datacenters. Thus, we call the Reduce task that writes its output to a remote datacenter as *datacenter-remote Reduce task*, and the Reduce task that writes its output only to the local datacenter as *datacenter-local Reduce tasks*.

The shuffle phase in the Hadoop MapReduce framework is an area where refinements and improvements are continually being made. During the shuffle phase, every Reduce task fetches its portion of intermediate data from each Map task's output file. Therefore, the shuffle phase generates large network I/O traffic and is the bottleneck of a MapReduce job. Shuffling data across datacenters can be more demanding on network I/O bandwidth and thus leads to even lower efficiency.

Figure 1 shows the execution of a MapReduce job WordCount in a sub-cluster. We make several interesting observations. First, Map tasks and Reduce tasks are executed in groups (*Map task group and Reduce task group*), the number of tasks in each group depends on the number of nodes in the cluster allocated for Map or Reduce tasks and the number of available slots on each node. The total number of Map tasks equals to the total number of input splits (80 in this example), while the total number of Reduce tasks is pre-configured (also 80 in this example). Second, by configuring early shuffle to start upon reaching 5% or more Map tasks completed, we observe that the shuffle phase of the first Reduce task group started upon the first Map task group finished. By taking a closer look, we see that although early shuffle enables the first Reduce task group to finish earlier, it does incur some delay in the execution of the other Map tasks that are concurrently running. In this example, the execution time of the following Map tasks is on average 23.9 seconds, longer than that in the first Map task group (20.9 seconds). Third, the average execution time of sort phase takes about 1 second. This is because the local combine operation at Map phase in WordCount decreases the size of intermediate data, which makes the sort phase in Reduce finishes very quickly. Fourth, when we zoom into each task group as shown in Figure 1, we find that the tasks in each single group do not start exactly at the same time because each task is encapsulated in a JVM, the time spent on initialization is not exactly the same for each JVM when they are competing for resources on the same physical machine. In the rest of the paper, the execution time of task groups is given by the average value.

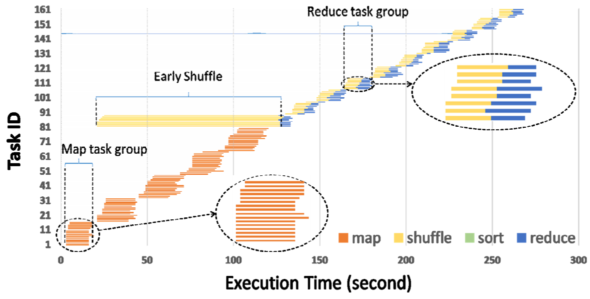


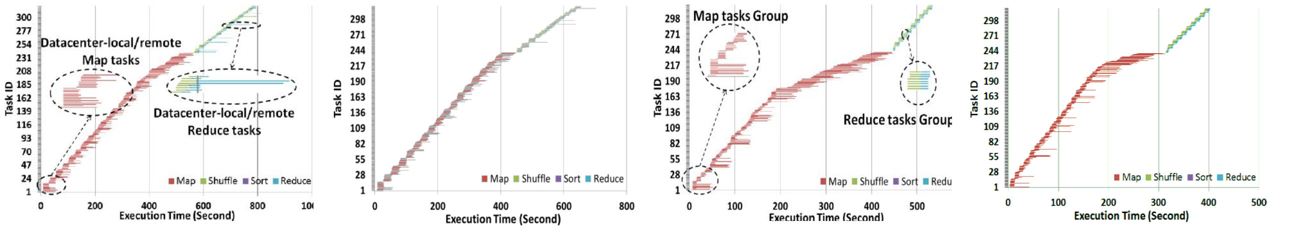
Figure 1. WordCount running in a sub-cluster

C. *Geo-Distributed Cluster: Design Consideration*

In order to gain a better understanding of performance bottlenecks of jobs running in a geo-distributed cluster, we conduct performance measurement study on a geo-distributed Hadoop cluster. We run MapReduce jobs on input data that are distributed in different sub-clusters in a simulated geo-distributed cloud, which consists of 20 nodes from two sub-clusters (*SCA* and *SCB*), and each has 10 nodes and is from different datacenters. The cross-datacenter access latency is 200ms, while the latency between the nodes within a single sub-cluster is about 1ms. .

Figure 2 shows how MapReduce jobs *WordCount* and *Grep* behave when they are running in our simulated geographically distributed cluster. X-axis shows the timeline of a job execution and Y-axis shows all task IDs in the order of their execution. We obtain four interesting observations. First, when zooming into a Map task group as shown in Figure 2(a), we notice that those Map tasks that run much longer are datacenter-remote Map tasks while those Map tasks that have relatively shorter execution time are the datacenter-local Map tasks. Also, the execution time of node-local Map tasks is very similar to that of the datacenter-local Map tasks when compare to the execution time of datacenter-remote Map tasks. Thus in the remaining analysis, we do not make explicit distinction between node-local Map tasks and datacenter-local Map tasks, and refer to both simply as datacenter-local Map tasks and compare them with datacenter-remote Map tasks.

Second, the ratio between datacenter-local Map tasks and datacenter-remote Map tasks is very similar to the ratio between local input data and remote input data. Consider WordCount with 15GB input data, the percentage of remote map tasks is 48% when 50% of its input data is from a remote datacenter (Figure 2(a)), and the percentage of remote map tasks decreases to 23% when 20% of its input data is remote (Figure 2(b)). Therefore, the percentage of



(a) WordCount on 15GB data, 50% input data remote

(b)WordCount on 15GB data, 20% input data remote

(c)Grep on 15GB data, 50% input data remote

(d)Grep on 15GB data, 20% input data remote

Figure 2. Execution of MapReduce jobs with distributed input data

remote input data may be used to estimate the percentage of datacenter-remote Map tasks in the subsequent discussion.

Third, we notice that, for *WordCount*, datacenter-remote Map tasks and datacenter-local Map tasks appear alternatively at the early stage of the Map phase. But as the Map phase progresses, the amount of datacenter-remote Map tasks increases significantly and dominates in the latter stage of the Map phase. In Figure 2(a), from 0 second to about 400 second, datacenter-remote Map tasks appear approximately every 25 seconds. However, there are very few datacenter-local Map tasks from 400 to the end of the Map phase. This is because at the beginning of the Map phase, the job will digest its input data from both datacenter-local and datacenter-remote input splits simultaneously as long as there are enough map slots. Since the datacenter-remote Map tasks take longer time to finish, the datacenter-local Map tasks will be finished faster. Given that 50% of the input data is distributed in the remote datacenter, thus at the latter stage of Map phase, the rest of Map tasks are dominated by datacenter-remote Map tasks. This remains to be true even when the percentage of local input data is more than that of remote input data, e.g., with 20% input data remote as shown n Figure 2(b), we still observe that the number of datacenter-remote Map tasks increases as the job approaches the latter stage of the Map phase.

Finally, when zooming into a Reduce task group as shown in Figure 2 for both *WordCount* and *Grep*, we find that the Reduce tasks with a much longer execution time are datacenter-remote Reduce tasks, while Reduce tasks with shorter execution time are datacenter-local Reduce tasks. It is interesting to note that the datacenter-remote Reduce tasks occur randomly. This is because the data placement policy in HDFS by default will choose three target nodes from the geographically distributed cluster, one of which is the node where the reduce write operation is issued, the other two are randomly chosen if no rack information is provided. If one of them is located in a different sub-cluster from the one where the write operation is issued, a datacenter-remote Reduce task will be created.

Figure 3 shows the Reduce phase of *WordCount* job running in the rack-aware HDFS configuration. We notice that the number of datacenter-remote Reduce tasks increases dramatically in this case. By the data placement policy of HDFS, when rack information is provided, the third replica will be stored in a node which is from a different rack. Therefore, a fair number of Reduce tasks become datacenter-remote Reduce tasks in this configuration. At the same time, the probability that all the replicas are stored in the same datacenter is not zero. Thus, datacenter-local tasks still exist. Similar results are also observed in executing the Grep jobs.

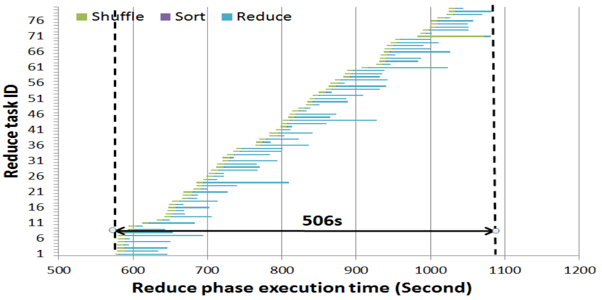


Figure 3. Reduce phase of WordCount

From the experimental analysis above, we argue that there are three main design considerations for provisioning a high performance geo-distributed cluster: First, mechanisms that can reduce the amount of intermediate data being shuffled across datacenters are critical for improving the performance of Hadoop deployment cross multi-datacenters. Second, adequate control of the number of datacenter-remote Reduce tasks can speed up the execution time of Reduce tasks. Third but not the least, mechanisms for reducing the amount of datacenter-remote Map tasks is necessary for effective Hadoop service provisioning in a geo-distributed cloud.

III. Our Solution Approach

Following the design guidelines outline in the previous section, we present our solution approach by focusing on the three optimization techniques, each targeting at one of the three problems: 1) *Sub-cluster aware job/task scheduling,* 2) *Sub-cluster aware reduce output placement,* and 3) *Map input data pre-fetching.*

Figure 4 shows the architecture of a geo-distributed cluster with our proposed solutions. It consists of sub-clusters from different datacenters, in which *Map phase exe time predictor, sub-cluster aware writer,* and *Map input pre-fetcher* are the core components that corresponding to our optimization techniques.

A. *Sub-cluster aware scheduling of jobs and tasks*

Given a job submitted to a geo-distributed cluster, the job scheduler should first examine whether the job can run

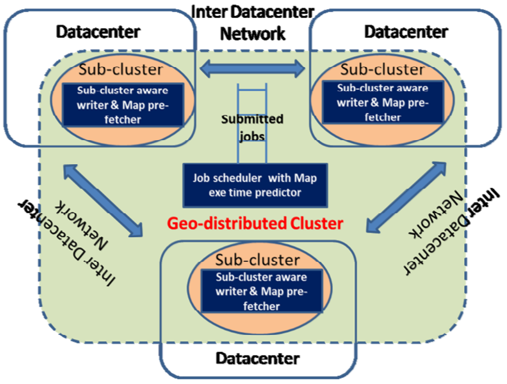


Figure 4. Architecture of a geo-distributed cluster

within a sub-cluster and which sub-cluster this job can run more efficiently. This optimization can significantly improve the performance of the shuffle phase by minimizing the unnecessary cross-datacenter network I/O traffics.

The objective of this optimization is to find a sub-cluster that can finish the given job in a shortest time, even if its input data is distributed in multiple local datacenters. We accomplish it by extending the current Hadoop MapReduce job scheduler and task scheduler. By making the scheduler sub-cluster aware, all the tasks of a given job can be assigned to the nodes within the same sub-cluster. Concretely, this optimization can be implemented in two steps: First, every submitted job should notify the job scheduler the name of the sub-cluster in which it prefers to running in. The job schedule can check if all the tasks of this job can be assigned to the nodes within this sub-cluster. Second, whenever receiving a task assignment request from a TaskTracker, the task scheduler checks which sub-cluster this TaskTracker belongs to, and only assign the pending Map tasks to the TaskTracker if it belongs to the same sub-cluster as the one chosen by the given job.

One challenge still exists to make this optimization work correctly and efficiently. Since it requires each MapReduce job to indicate its preferred sub-cluster, we extend the job scheduler by adding the prediction capability that can estimate which sub-cluster this job can be executed in the shortest time. When all input data of a job are located within one sub-cluster, it is relatively easy to estimate the job execution time of such sub-clusters and choose the one with the shortest execution time. However, the input data of a job may not be all located in the same sub-cluster. We address this challenge by providing a *Map phase execution time prediction model* below.

Map phase execution time prediction. When a MapReduce job is submitted to a geographically distributed cluster, its input data could be either located in a single sub-cluster, or distributed among multiple sub-clusters. Considering the heterogeneous capability among different sub-clusters, simply putting the job in the sub-cluster, which has the largest portion of its input data may not be optimal. Therefore, we propose a prediction model to estimate the Map phase execution time of a particular job running in a given sub-cluster.

We present our prediction model for Map task execution time based on two scenarios: *(i) local input data* and *(ii) distributed input data*. In the first case, all the Map tasks are datacenter-local Map tasks, while in the second case, there exist both datacenter-local Map tasks and datacenter-remote Map tasks. Before discussing the prediction model, we present the list of parameters and terms we will use.

- **Execution time of Map Phase (EMP):** The total execution time of the Map phase in a MapReduce job. It measures the duration from the beginning of a MapReduce job to the end of the last group of its Map tasks.
- **Execution time of a Datacenter-Local Map task (EDLM):** The average execution time of datacenter-local Map tasks.
- **Execution time of a Datacenter-Remote Map task (EDRM):** The average execution time of datacenter-remote Map tasks.
- **Percentage of Remote input data (PR):** The portion of the input data that is located in a different sub-cluster from the one in which the MapReduce job is running in.
- **Number of Map Tasks in the job (NMT):** The total number of Map tasks to run in a MapReduce job.
- **Number of Map Slots in a sub-cluster (NMS):** The total number of Map slots that can be used by a MapReduce job in a given sub-cluster.

Among all the above parameters, the value of EMP is what we are going to predict. The value of EDLM and EDRM are gathered from historical execution statistics of similar MapReduce jobs. The value of PR, NMT, and NMS can be obtained from cluster configuration.

(1) Local input data

This case is the easiest and most straightforward one. Figure 5 shows an abstraction of Map phase execution based on previous experiments. Each yellow bar represents a group of Map tasks that are executing simultaneously. The length of the yellow bar reflects *ELDM*, while the width of the yellow bar represents *NMS*. Since all the Map tasks are datacenter-local Map tasks in this case, their execution times are very much similar to each other when there are no stragglers. Therefore, the execution time of the whole Map phase can be calculated as following:

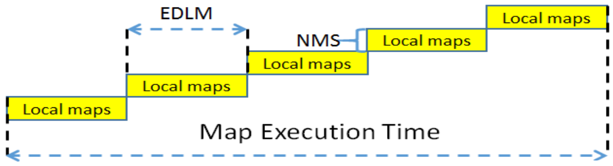


Figure 5. Abstraction of Map phase in case (1)

$$EMP = ELDM \cdot \frac{NMT}{NMS}$$

(2) Distributed input data

Figure 6 shows the abstraction of Map phase execution in the case of distributed input data. Both Datacenter-local Map tasks and datacenter-remote Map tasks will be scheduled simultaneously at the beginning of the job. The yellow bars, which are shorter, represent the execution of datacenter-local

Map tasks. The blue bars, which are longer, represent the execution of datacenter-remote Map tasks. Though the execution time of a datacenter-remote Map task is set to be twice as long as that of a datacenter-local Map task in Figure 6, the ratio between the execution time of these two categories of Map tasks can vary in general.

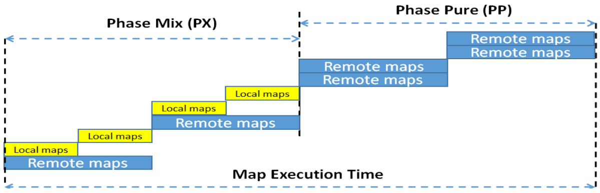


Figure 6. Abstraction of Map phase in case (2)

According to Figure 6, the whole Map phase is divided into two stages. *Phase Mix* represents the time that datacenter-remote Map tasks execute simultaneously with datacenter-local Map tasks. *Phase Pure* represents the execution of datacenter-remote Map tasks only. Therefore, the execution time of Map Phase is the summation of the *execution time of Phase Mix (PM)* and the *execution time of Phase Pure (PP).*

$$PM = EDLM \bullet \frac{NMT(1-PR)}{NMS(1-PR)} = \frac{NMT \bullet EDLM}{NMS}$$

As mentioned earlier, we use input data distribution to estimate the Map tasks distribution. Therefore, $NMT(1-PR)$ represents the total number of datacenter-local Map tasks. $NMS(1-PR)$ equals to the number of slots that can be used to run datacenter-local Map tasks in Phase Mix. Then, the execution time of Phase Pure is calculated as:

$$PP = EDRM \bullet \left\lceil \frac{NMT \bullet PR - NMT \bullet (1-PR) \bullet \frac{EDLM}{EDRM}}{NMS} \right\rceil$$

In which $NMT \bullet PR$ represents the total number of datacenter-remote Map tasks, and $NMT \bullet (1-PR) \bullet EDLM / EDRM$ is the number of finished datacenter-remote Map tasks in Phase Mix. Therefore

$$EMP = PM + PP$$

B. *Sub-cluster Aware Reduce Output Placement*

As we mentioned earlier, HDFS uses replication to provide data availability in case of node failure. By default, all HDFS files are stored in three replicas. Given that the reduce output file is to be written directly through HDFS, the selection of target nodes can affect the execution time of the Reduce tasks. If all the selected target nodes are from the same sub-cluster in which the Reduce task is running, this Reduce task will be finished faster. Otherwise, at least one of the target nodes is selected from a different datacenter, and this Reduce task will take a much longer time to finish, since it has to flush data to another datacenter.

Current HDFS's data placement policy is not sub-cluster aware. In order to avoid data flushing between different sub-clusters and make all Reduce tasks as datacenter-local Reduce tasks, we modify the data placement policy in HDFS as follows: First, HDFS's NameNode should be able to distinguish different sub-clusters. At the registration stage, each DataNode sends the name of the sub-cluster to which it belongs to the NameNode, so that NameNode is able to obtain the structure of the whole geo-distributed cluster. Second, HDFS's NameNode should select all the target nodes from the same sub-cluster as the one from which the Reduce write operation is issued. This guarantees that there will be no datacenter-remote Reduce task.

One of our ongoing research efforts is to introduce periodic availability-based replication to meet desired availability demand of cloud consumers. In addition, we will extend the load balancing check to make it sub-cluster aware.

C. *Map Input Data Pre-fetching*

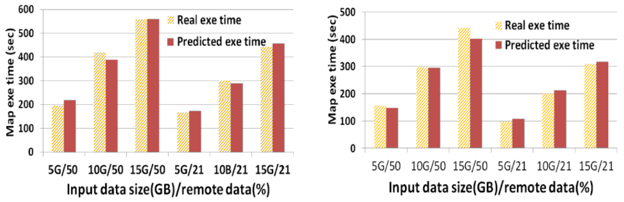
This optimization strategy is to eliminate the amount of datacenter-remote Map tasks by incorporating pre-fetching of map input data. As shown in Figure 2, if the input data of a MapReduce job is scattered around multiple datacenters, datacenter-local Map tasks will execute simultaneously with datacenter-remote map tasks at the beginning the Map phase. This provides the opportunity to further improve the performance of Map phase by interleaving the processing of local input data and the pre-fetching of remote input data. Thus all the Map tasks could be executing locally without accessing data from a remote sub-cluster.

To implement this optimization, we need to make two additional modifications to the task scheduler. First, we need to assign the processing of local input data with a higher priority than processing the remote input data. Since the Map task scheduler is already sub-cluster aware, it should be able to schedule the Map tasks to process the input data from local sub-cluster first. This implies that at the beginning of the Map phase, all the Map tasks will be datacenter-local Map tasks. Second, we need to implement the interleaving of the processing of local input data with the pre-fetching of remote input data. While processing the local input data at the beginning of the Map phase, the job scheduler notifies the TaskTrackers to fetch the input data from remote sub-clusters to the local-cluster. Therefore, after processing all the pre-exist local input data, remote input data is now also available in the local sub-cluster, and datacenter-remote Map tasks can be eliminated.

IV. EXPERIMENTS

In this section, we present experimental results evaluating the effectiveness of our above mentioned optimizations for MapReduce jobs that are running in a geo-distributed cluster.

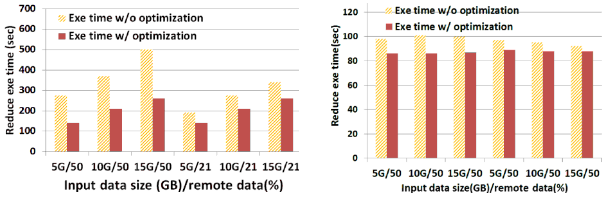
The results can be divided into two categories. First, we show the accuracy of the Map phase execution time prediction model by running representative MapReduce jobs on various distribution of input data. Second, we plot how much performance of Reduce phase can be improved by utilizing *Sub-cluster aware reduce output placement.* The setup of experimental environment is the same as those specified in Figure 2.



(a) Map exe time prediction for WordCount

(b)Map exe time prediction for Grep

Figure 7. Accuracy of the prediction model



(a) Reduce exe time of WordCount

(b) Reduce exe time of Grep

Figure 8. Reduce phase performance improvement

In the first set of experiments, *WordCount* and *Grep* have been used as representative MapReduce jobs that are running in the geographically distributed cluster. The aim of these experiments is to see the accuracy of our prediction model. Therefore, we first calculate the Map phase execution time using the prediction model assuming that the job is going to run in *Sub-Cluster A (SCA)*. Then, we actually ran the job in *SCA*, and measure the real execution time of the Map phase. After that, we compare the predicted value and the measured value to see the accuracy of the prediction model. In order to make the results more convincing, each job is going to run multiple times with various amount of input data as well as different forms of input data distribution, so that we can see whether the prediction model works well in various situations.

Figure 7 shows the results from this set of experiments. All the jobs are running in *SCA* but with different amount of total input data and different forms of input data distribution. For example, '20% data remote' means when this job starts running in *SCA*, while 20% of its input data is located in *SCB* and 80% of them is located in *SCA*. From this figure, we observe that first, the more input data is located in *SCB*, the longer the Map phase execution time is. This is because larger amount of remote input data leads to more datacenter-remote Map tasks, which degrades the performance of the whole Map phase. Second, the predicted value of Map phase execution time can be either longer or shorter than the real value. However, the error is within 5%, which is acceptable.

The second set of experiments shows the improvement of Reduce phase performance by utilizing *Sub-cluster aware reduce output placement* optimization. Similar to the experiments in the first set, two representative categories of MapReduce jobs, *WordCount* and *Grep*, are running in a simulated geographically distributed cluster with *Sub-cluster aware reduce output placement* enabled and disabled respectively. The execution time of the whole Reduce phase is recorded and compared as an indicator of the performance.

Figure 8 shows the results of comparing the execution time of the Reduce phase between different jobs running in the simulated geographically cluster with our optimization enabled and disabled. The configurations of these experiments are the same as those shown in Figure 2. An interesting observation from this figure is that, in all the tested cases, by enabling *Sub-cluster aware reduce output placement*, the average Reduce phase execution time has been reduced up to 48% for the *WordCount* job, while that of *Grep* job has only reduced by 13%. In other words, this optimization brings larger extent of improvement to *WordCount* than to *Grep*. This is because in our experiments, the Reduce output data of the job *WordCount* is larger than that of *Grep*. For example, *WordCount* job with 15GB input, 50% of which is remote, writes 1151.89MB of data back through HDFS, whereas *Grep* job in the same case only writes back 8432 Bytes of data. And this optimization mainly works on accelerating the write back of the Reduce output through HDFS. Therefore, the more data the Reduce tasks need to write back, the larger extent they can be benefited.

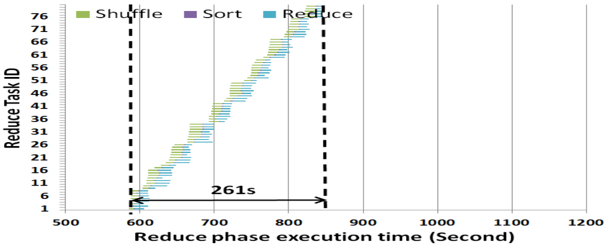


Figure 9. Reduce phase of WordCount with optimization enabled

Figure 9 shows the detailed execution of the Reduce phase from a *WordCount* job that is running in a geo-distributed cluster with *Sub-cluster Aware Reduce Output Placement* enabled. The amount of input data for this job is 15GB, 50% of which is located in a remote sub-cluster. Each horizontal line in the figure represents the execution of a Reduce tasks. Recall that in Figure 3, the length of some lines are longer than that of the others, which means the execution time of each Reduce tasks varies from each other. As we analyzed earlier, this is because the co-existence of datacenter-local Reduce tasks and datacenter-remote Reduce tasks. With these two categories of Reduce tasks running simultaneously, the execution time of the Reduce phase shown in Figure 3 is 506 seconds. In Figure 9, however, we observe that the execution time of all Reduce tasks is similar to each other. In the other words, all the Reduce tasks become datacenter-local Reduce tasks. That is because the modified HDFS data placement policy choose to put all the replicas into the same sub-cluster where the write operation is issued, therefore datacenter-remote Reduce tasks are eliminated. With this optimization, we observe that the execution time of the whole Reduce phase reduces 48.4% from 506 seconds to 261 seconds.

V. Related Work

Facebook's project Prism [12] is a project that supports geographically distributed Hadoop data stores, which removes the limitations on Hadoop's capacity to crunch data. With Prism, a logical abstraction layer is added so that Hadoop cluster can run across multiple data centers, effectively removing limits on capacity. However, Prism has not been open-sourced, neither the technique details of Prim has been released. In addition, Google built and deployed its globally-distributed relational database management system Spanner [6], which automatically migrates data across machines or even across datacenters to achieve load balancing and failure recovery. Cisco and HP are also making efforts to connect geographically dispersed datacenters [13, 14] by optimizing the data link layers of the network among datacenters. However, current interconnect methods suffer from limitations such as transport dependency, complexity and lack of resiliency. Since we focus on connecting datacenters from software layer, our solution approach can integrate with their solution to provide highly efficient and scalable geo-distributed cloud.

Lots of efforts from academia in recent five years are dedicated to better utilize the capacity of geo-dispersed datacenters to improve the performance of applications running on top of them. Using geographically load balancing, [7] investigates the opportunities of lowering the operational cost of each datacenter. [8, 9, 10] are working on evaluating and improving the performance of MapReduce jobs that are running on highly distributed data. [8] categories MapReduce jobs into three types: local, global, distributed, and analyzes the performance of these three types of jobs. [11, 16] propose a hierarchical MapReduce to alleviate the performance degradation in distributed environment. In comparison, our work is the first to provide comprehensive study and optimizations for improving Hadoop service provisioning in multi-datacenter Hadoop Cloud.

VI. Conclusion

We have described our approach to provisioning high performance Hadoop clusters in a geographically distributed cloud. We first compare multi-datacenter Hadoop deployment with single-datacenter Hadoop deployment to identify the performance issues inherent in a geographically distributed cloud. Then we describe a suite of system-level optimizations for improving performance of Hadoop service provisioning in a geographically distributed cloud, including prediction-based MapReduce job localization and task scheduling, sub-cluster aware Reduce output placement, and Map input data pre-fetching. We evaluate our solution through experimental evaluation and show that our prediction based localization has very low error ratio, smaller than 5%, and our optimization can improve the execution time of Reduce phase by 48.6%.

VII. Acknowledgement

This research is partially supported by grants from NSF CISE NetSE program, SaTC program, I/UCRC, an IBM faculty award and a grant from Intel ICST on Cloud Computing.

References

[1] K. Rajamani, C. Lefurgy, S. Ghiasi, J. Rubio, H. Hanson, and T. Keller. "Power management solutions for computer systems and datacenters." In Low Power Electronics and Design (ISLPED'08), ACM/IEEE International Symposium on, Bangalore, India, 2008, pp. 135-136..

[2] J. Baker, C. Bond, J. Corbett, J. Furman, A. Khorlin, J. Larson, J.M. Leon, Y. Li, A. Lloyd, and V. Yushprakh: "Megastore: Providing scalable, highly available storage for interactiveservices." In: Conf. Innovative Data Systems Research (CIDR'11), Asilomar, California, 2011, pp. 223–234

[3] M. Burrows. "The chubby lock service for loosely-coupled distributed systems." In: Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI'06), Seattle, USA, 2006 ,pp. 335–350.

[4] T. Kraska, G. Pang, M.J. Franklin, S. Madden, and A. Fekete. "Mdcc: Multi-data center consistency." In Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys'13), Prague, Czech Republic, pp. 113-126

[5] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. "Dynamo: amazon's highly available key-value store." In Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP'07), Stevenson, USA, 2007, vol. 7, pp. 205-220.

[6] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. "Spanner: Google's globally-distributed database". In Proceedings of the 10th symposium on Operating Systems Design and Implementation (OSDI'12), HOLLYWOOD, USA, 2012, pp. 251-264

[7] H. Goudarzi, and M. Pedram. "Geographically Load Balancing for Online Service Applications in Distributed Datacenters." In Proceedings of the IEEE 6th International Conference on Cloud Computing (CLOUD'13), San Jose, USA, 2013, pp. 351-358

[8] M. Cardosa, C. Wang, A. Nangia, A. Chandra, and J. Weissman. "Exploring mapreduce efficiency with highly-distributed data." In Proceedings of the 2nd international workshop on MapReduce and its applications (MapReduce'11), ACM ,San Jose, USA, 2011, pp. 27-34.

[9] A. Mandal, Y. Xin, I. Baldine, P. Ruth, C. Heerman, J. Chase, V. Orlikowski, and A. Yumerefendi. "Provisioning and evaluating multi-domain networked clouds for hadoop-based applications." In Proceedings of IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom'11), Athens, Greece, 2011, pp. 690-697.

[10] L. Wang, J. Tao, H. Marten, A. Streit, S.U. Khan, J. Kolodziej, and D. Chen. "MapReduce across distributed clusters for data-intensive applications." In IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW'12), Shanghai, China, 2012, pp. 2004-2011.

[11] T. Bicer, D. Chiu, and G. Agrawal. "A framework for data-intensive computing with cloud bursting." In Proceedings of IEEE International Conference on Cluster Computing (CLUSTER'11), Austin, Texas, 2011

[12] Facebook's Prism Project
http://www.wired.com/wiredenterprise/2012/08/facebook-prism/

[13] "Interconnecting Geographically Dispersed Data Centers Using VPLS-Design and System Assurance Guide", Cisco Systems, Inc, USA, 2009

[14] "Connecting Geographically Dispersed Data Centers", HP

[15] R.D. Prisco, B. Lampson, and N. Lynch. "Revisiting the PAXOS algorithm." Theoretical Computer Science 243, no. 1 (2000): 35-91.

[16] Y. Luo, Z. Guo, Y. Sun, B. Plale, J. Qiu, and W.W. Li. "A hierarchical framework for cross-domain MapReduce execution." In Proceedings of the second international workshop on Emerging computational methods for the life sciences, pp. 15-22. ACM, 2011.