Improving Hadoop Performance in Intercloud Environments

Shin-gyu Kim*, Junghee Won†, Hyuck Han*, Hyeonsang Eom*, Heon Y. Yeom*
*Seoul National University, †SAP Korea Ltd.
*{sgkim, hhyuck, hseom, yeom}@dcslab.snu.ac.kr, †jeong.hee.won@sap.com

ABSTRACT

Intercloud is a federated environment of private clusters and public clouds. The performance of Hadoop could be degraded significantly in intercloud environments. Because previous solutions for intercloud environments rely on speculative execution, they require additional cost in the cloud. In this paper, we propose a new task scheduler that improves performance without the help of speculative execution in intercloud environments.

1. INTRODUCTION

Hadoop continues to increase in its popularity in large scale data processing because of its scalability and fault tolerance. When they need more computational power for heavy workloads, the capability of the Hadoop platform can be easily increased by just joining additional compute nodes. However, it is expensive to acquire and maintain large-scale clusters for unexpected heavy workloads. One of the affordable options is renting compute nodes from cloud computing services. The flexible pay-as-you-go pricing model of the cloud could help reduce the investment in private cluster infrastructures.

Hadoop implicitly assumes that compute nodes are homogeneous. However, if additional compute nodes join from the cloud, the above assumption does not hold any longer. We call this environment an intercloud environment. It is widely known that Hadoop's performance can be degraded significantly in intercloud environments [4, 1]. There are several studies designed to solve this problem [4, 2, 1], and most of them rely on speculative execution. It is noted that speculative tasks are also charged in the cloud, and they could elongate the execution time of other tasks by competing for scarce resources, such as the network.

In this paper, we propose ICMR (MapReduce for InterCloud environments) to solve the above problem without speculative execution. The fundamental idea is that ICMR dynamically adjusts workload for each compute node in proportion to their processing speed. In intercloud environments, input data is stored only in private clusters, and the compute nodes from the cloud read input from the private clusters. Because map workers can make progress with reading input data simultaneously, it is not necessary to copy or move input data to the cloud prior to the start of map tasks. In order to reduce total running time, the reduce phase should be started as soon as possible [3]. For this reason, ICMR aims to make all map worker nodes finish at the same time.

2. HADOOP FOR INTERCLOUD ENVIRONMENTS

2.1 Execution Model of Hadoop

In this section, we present the execution model of Hadoop. For simplicity, we assume that all map workers start at the same time. Because the reduce phase is not controllable by the task scheduler, we omit the reduce phase in explanation. Figure 1 illustrates the execution model of Hadoop with two worker nodes, and M and S mean map phase and shuffling phase, respectively. Node 1 and node 2 represent the private clusters and compute nodes from the cloud, respectively. In node 1, map processing time is longer than shuffling time, and in node 2, shuffling time is longer than map processing time due to the slow wide-area network.

Total running time of node i (T_i) is composed of total map processing time (M_i), total shuffling time (S_i) and overlapped time between M_i and S_i. The total map processing time (M_i) is as follows:

$$M_i = \frac{W_i}{v_i}, \quad v_i = min(fs_i,\ p_i) \tag{1}$$

where W_i is the amount of input data for node i, and v_i is

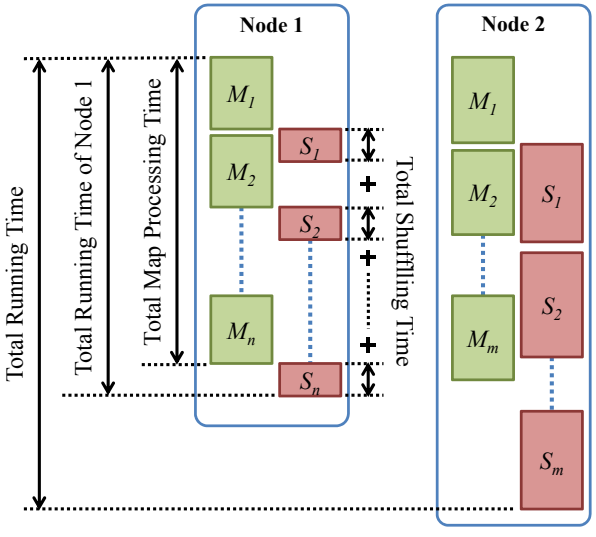

Figure 1: Execution model of Hadoop, including map and shuffle phases.

processing speed of node i. The processing speed v_i is determined by two performance metrics, which are data reading rate from storage (fs_i) and the computing power of node i (p_i). In compute nodes from the cloud, fs_i has a lower value than p_i. The total map shuffling time (S_i) is as follows:

$$S_i = \frac{I_i}{t_i/N_i} \quad (2)$$

where I_i is the amount of intermediate data produced at node i, and t_i is the data transfer rate to reduce workers. Because shuffling can happen only one at a time in each worker node, t_i is divided by the number of map workers in node i (N_i). From the above equations, we get a total running time of node i (T_i):

$$T_i = M_i + S_i - \text{overlapped time} \quad (3)$$

If the total running time is not short enough, the overlapped time is almost same as M_i or S_i depending on the execution environment. Thus, T_i can be represented as follows:

$$T_i \approx max(M_i,\ S_i) \quad (4)$$

Finally we get a total running time for the whole job (T) as below.

$$T = max(T_1, \ldots, T_n) \quad (5)$$

2.2 ICMR Scheduler

The objective of the ICMR scheduler is to finish shuffle phases of all map workers at the same time. By doing this, idle resources are reduced, and the reduce phase can be started as soon as possible. Because the original Hadoop scheduler (the left part in Figure 2) does not consider the performance of compute nodes and the network, the end times of map workers could be different. However, our ICMR scheduler (the right part in Figure 2) adjusts the amount of input data (W_i) for each compute node in proportion to its processing speed (v_i), and achieves the goal.

We assume the amount of intermediate data (I_i) is proportional to the amount of input data (W_i).

$$I_i = \alpha \times W_i \quad (6)$$

In equation 6, α is the ratio between I_i and W_i. α is measured and updated whenever one map task is completed. The total amount of input data (W_{total}) is invariant as below.

$$W_{total} = W_1 + W_2 + \ldots + W_n \quad (7)$$

If all map workers finish shuffle phases at the same time, we get the following relation from equations 1, 2, 4, and 6.

$$max\left(\left(\frac{W_1}{v_1}\right), \left(\frac{\alpha W_1}{t_1/N_1}\right)\right) = \ldots = max\left(\left(\frac{W_i}{v_i}\right), \left(\frac{\alpha W_i}{t_1/N_i}\right)\right)$$

From the above relation, we can derive W_i ($1 \le i \le n$).

$$\left(\frac{W_1}{v_1}\right) = \ldots = \left(\frac{\alpha W_j}{t_j/N_j}\right) = \left(\frac{W_k}{v_k}\right) = \ldots$$

$$W_j = \left(\frac{W_1}{v_1} \times \frac{\alpha}{t_j/N_j}\right),\ W_k = \left(\frac{W_1}{v_1} \times v_k\right), \ldots \quad (8)$$

Because v_i, α, t_i and N_i can be measured at the early time of a job, we can calculate W_1 from equations 7 and 8.

ICMR dynamically adjusts W_i in proportion to the performance of each compute node. First, ICMR assigns equal amount of task to all compute nodes, because it has no measured values for essential metrics of α, v_i and t_i. After several tasks are finished, the central job tracker of Hadoop gathers initial values for the metrics, and ICMR can calculate W_i for all compute nodes. During processing of jobs, the performance of network and CPU would be varied. Therefore, the performance metrics which include α, v_i, t_i, are measured and updated periodically. Then, ICMR dynamically adjusts the amount of workloads by recalculating all W_i.

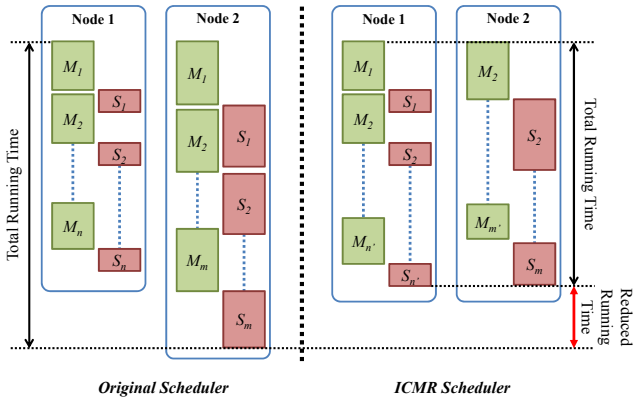

Figure 2: Overview of the ICMR scheduler which considers shuffling time as well as map processing time.

The original Hadoop scheduler cuts up input data into equal chunks. However, we modified Hadoop to be able to have different sizes of chunks. This modification plays an important role at the last phase. As shown at the last phase of node 2 in the ICMR scheduler part of Figure 2, the last chunk is smaller than the former chunks. If the size of the last data chunk is equal to other chunks, the running time of node 2 is increased and the start of the reduce phase should be delayed. Although the amount of managing information of ICMR scheduler is increased, we modified Hadoop to improve overall performance.

2.3 Preliminary Result

We implemented the prototype of ICMR in Hadoop 0.19.2, and conducted an experiment on Amazon EC2. We used four small instances from U.S. west and east each, and the total number of compute nodes was eight. The small instance of Amazon EC2 has one virtual CPU and 1.7 GB of memory, and the network bandwidth between the two regions was about 70 Mbps. We executed a wordcount application with 10GB data. The data was stored only at U.S. east, and the replication factor of Hadoop file system (HDFS) was three. We configured Hadoop to have 4 map workers at U.S. east and west each, and 4 reduce workers only at U.S. east.

Figure 3 shows the experimental result. The left bar is for the original Hadoop scheduler, and the right bar is for our ICMR scheduler. In this experiment, ICMR reduced total map and shuffling time by about 19%. Because all compute nodes had almost same performance, the difference was due to the limitation of the network bandwidth between U.S. east and U.S. west clouds. If a data consuming rate of a

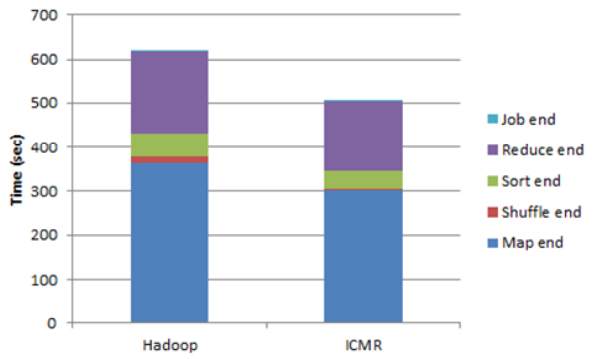

Figure 3: Performance comparison between original Hadoop scheduler and ICMR scheduler when executing a wordcount application.

map worker is higher than a data transfer rate, the map worker should wait for the next data without progress. In this experiment, the data transfer rate was slower than the data consuming rate. Because the processing speed of node v_i in our ICMR scheduling model measures not a CPU clock rate but data processing speed, the total running time of map processing time was also decreased.

3. CONCLUSION

In this paper, we proposed an ICMR scheduler that improves Hadoop performance in intercloud environments without the help of speculative execution. In future work, we will evaluate ICMR with various workloads on large scale clusters and the clouds. In addition to this, we are also planning to design a new cloud instance scheduler for Hadoop.

4. ACKNOWLEDGMENTS

This work was supported by Mid-career Researcher Program through NRF grant funded by the MEST (No. 2011-0018022). The ICT at Seoul National University provided research facilities for this study.

5. REFERENCES

[1] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris. Reining in the outliers in map-reduce clusters using mantri. In *Proceedings of OSDI 2010*.

[2] H. Lin, X. Ma, J. Archuleta, W.-c. Feng, M. Gardner, and Z. Zhang. Moon: Mapreduce on opportunistic environments. In *Proceedings of HPDC 2010*.

[3] A. Verma, N. Zea, B. Cho, I. Gupta, and R. H. Campbell. Breaking the mapreduce stage barrier. In *Proceeding of CLUSTER 2010*.

[4] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In *Proceedings of OSDI 2008*.