International Journal of Grid and Distributed Computing
Vol.6, No.5 (2013), pp.71-82
http://dx.doi.org/10.14257/ijgdc.2013.6.5.07

Improving MapReduce Performance by Data Prefetching in Heterogeneous or Shared Environments

Tao Gu, Chuang Zuo, Qun Liao, Yulu Yang and Tao Li

The College of Information Technical Science, Nankai University, Tianjin 300071, China

{gutao, zuochuang, liaoqun}@mail.nankai.edu.cn, {yangyl, litao}@nankai.edu.cn

Abstract

MapReduce is an effective programming model for large-scale data-intensive computing applications. Hadoop, an open-source implementation of MapReduce, has been widely used. The communication overhead from the big data sets' transmission affects the performance of Hadoop greatly. In consideration of data locality, Hadoop schedules tasks to the nodes near the data locations preferentially to decrease data transmission overhead, which works well in homogeneous and dedicated MapReduce environments. However, due to practical considerations about cost and resource utilization, it is common to maintain heterogeneous clusters or share resources by multiple users. Unfortunately, it's difficult to take advantage of data locality in these heterogeneous or shared environments. To improve the performance of MapReduce in heterogeneous or shared environments, a data prefetching mechanism is proposed in this paper, which can fetch the data to corresponding compute nodes in advance. It is proved that the proposal of this paper reduces data transmission overhead effectively with theoretical analysis. The mechanism is implemented and evaluated on Hadoop-1.0.4. Experiment results on real applications show that the data prefetching mechanism can reduce data transmission time by up to 94%.

Keywords: *MapReduce, Hadoop, Data Prefetching, Data Transmission*

1. Introduction

As the development of social networks, bio-computing, experimental nuclear physics and Internet of Things in recent years, massive data are produced. The need of big data processing is increasing rapidly. Representative data-intensive applications such as data mining and web indexing are becoming more and more important in both industry and academia. Achievements in big data research impact the development of both economic and society powerfully.

MapReduce is a feasible and scalable programming model for data-intensive computing applications [1-3]. Hadoop, an open source implementation of MapReduce framework, has been adopted by hundreds of institutions and companies including Yahoo, Facebook and Amazon [4, 5]. Hadoop is also used to support data-intensive scientific analysis such as data mining [6] and bioinformatics [7].

Data transmission of data-intensive applications is an important factor impacting the efficiency of MapReduce. To fully utilize the computing resource of a cluster, MapReduce deploys tasks to all nodes in the cluster, thus it's possible that a task needs to process data not stored locally. Noticing the overhead of data transmission, Hadoop tries to decrease the impact of data transmission by preferentially scheduling tasks to nodes storing corresponding

ISSN: 2005-4262 IJGDC

data. In consideration of data locality in scheduling, Hadoop works well in homogeneous and dedicated environments. However, in practice, clusters composed of commodity computers are usually heterogeneous and shared by multiple users for the resource utilization and cost limitation. Unfortunately, it is difficult for Hadoop to achieve good data locality in heterogeneous or shared environments [8-10].

To decrease the performance degradation caused by data transmission, a data prefetching mechanism is proposed in this paper. In the proposal, a data prefetcher is designed for keeping fetching input data from remote node. In this way, data prefetching is carried out concurrently with data processing, thus data transmission is overlapped with data processing in the time dimension. By fetching input data to corresponding compute nodes in advance, the execution time of a MapReduce job can be reduced effectively. Theoretical analysis and experimental result prove the performance of MapReduce is improved efficiently by the proposed data prefetching mechanism.

The rest of this paper is organized as follows. Section 2 states background and motivation. In section 3, the design and theoretical analysis of data prefetching mechanism is presented. Experiments and evaluation are given in Section 4. Section 5 introduces major related work, and conclusions and future work are given in the last section.

2. Background and Motivation

2.1. MapReduce overview

MapReduce adopts a divide-and-conquer approach for data-intensive applications. In MapReduce model, the execution of an application can be divided in two phases: map and reduce, as illustrated in Figure 1. In the map phase, amounts of map tasks process data blocks independently. After all map tasks are finished, the reduce phase begins. The intermediate results of map tasks are shuffled, sorted and then processed in parallel by one or more reduce tasks.

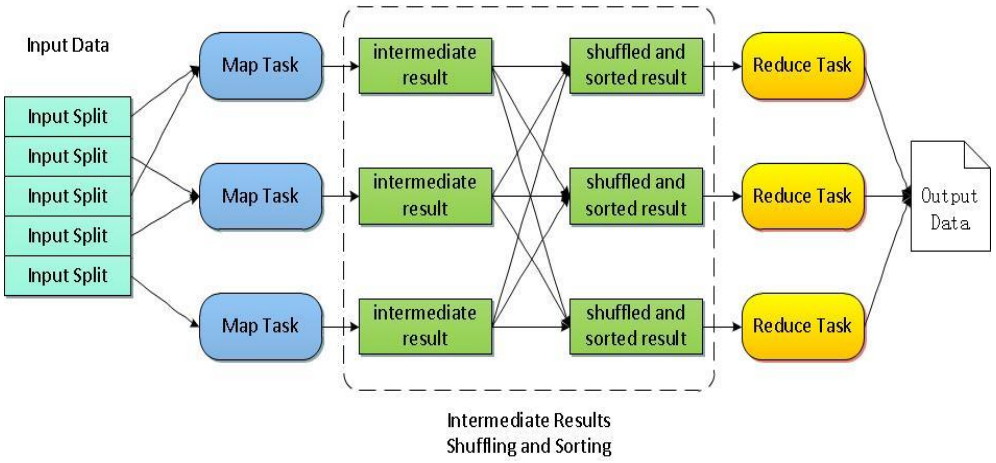

Figure 1. The workflow of MapReduce framework

Hadoop implements MapReduce framework in a master/slave architecture, which is composed of one master node and multiple slave nodes. The master node called JobTracker is in charge of job scheduling and cluster management, and slave nodes called TaskTracker are responsible for executing map and reduce tasks. The JobTracker schedules map and reduce tasks to appropriate nodes and keeps monitoring the state of tasks and nodes during the

execution of jobs. When a TaskTracker gets a task from JobTracker, it fetches corresponding input data, executes the task and returns the location of result to JobTracker. It is worth noting that the input data of map tasks and the result of reduce tasks are stored in Hadoop Distributed File System (HDFS)[11], which is a distributed file system implemented in Hadoop.

2.2 Problem statement and motivation

As described in Section 1, in heterogeneous or shared environments, the overhead of data transmission affects the performance of MapReduce greatly. This paper focuses on the data transmission in map tasks, which is discussed in detail below.

In the implementation of Hadoop, the input data of map tasks is not transferred all in one block, but in a mass of small slices instead. As a map task obtains a slice of the input data, it starts processing received data. When the processing of this slice of data finishes, the map task begins to transfer and process another data slice. This procedure repeats until the map task finishes processing all input data. The data transmission and data processing happens in sequence, so the overhead of data transmission cannot be avoided. Figure 2 illustrates the whole processes of a map task.

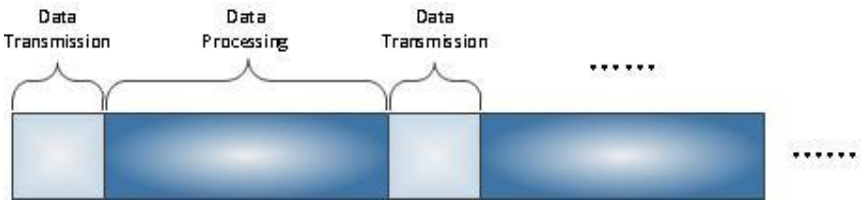

Figure 2. Alternation of data transmission and data processing in a map task

A map task can be regarded as a series of data processing and data transmission procedures. In Hadoop, the two procedures are serial. That means when the data is processed in a map task, no data transmission is carried out. As a result, the data processing has to wait for the next data slice when it is transferred. An improvement is inspired by overlapping the data transmission process with the data processing process to hide the overhead of data transmission effectively. Thus a data prefetching mechanism is proposed from this motivation.

3. Proposed Data Prefetching Mechanism

The basic idea of data prefetching mechanism proposed in this paper is to overlap the data transmission process with the data processing process. In this way, when the input data of map tasks is not local, the overhead of data transmission is hidden. As a result, the overall performance of MapReduce can be improved.

3.1. Data prefetching's design

The process of proposed data prefetching mechanism is shown in Figure 3: A map task M is assigned to node N1 but the nearest replication of M's input data is located in node N2, so N1 has to fetch data from N2 for following map processing. A data prefetching thread is created for requesting corresponding input data and a buffer called prefetching buffer is allocated on node N1 for storing fetched data temporarily.

In the execution of a map task, shown in the Figure 3, after the node N1 begins executing map task M, the data prefetching thread acquires the location of input data, which is node N2.

Then data prefetching thread keeps fetching input data from node N2 to prefetching buffer. And the map task M keeps processing input data in prefetching buffer. Obviously, this process is one application of typical producer-consumer model, where the data prefetching thread is the producer and the map task is the consumer. In the execution of a job, when any node gets a task whose input data is not local, the above described process is triggered for data prefetching.

One issue needed to be explained is the size of prefetching buffer. Intuitively, bigger prefetching buffer means better performance. But in fact, according to the producer-consumer model, two buffer units are enough. So the proposed prefetching mechanism needs little extra memory.

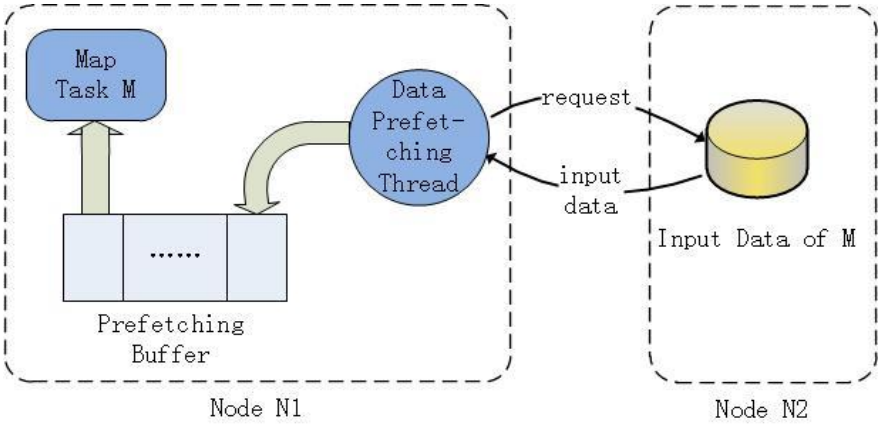

Figure 3. Data prefetching mechanism for Hadoop

3.2. Theoretical analysis

Compared with the native Hadoop, the improvement of improved Hadoop with proposed data prefetching mechanism is analyzed below. To do that, following definitions are given:

S_{map}: size of input split, i.e., the input data size, of a map task;

S_{buf}: size of buffer used in Hadoop for storing a slice of input data temporarily;

V_{map}: size of data processed by a map task per second in native Hadoop;

V'_{map}: size of data processed by a map task per second in improved Hadoop;

V_{tran}: size of data transferred per second between the node executing a map task and the node owning corresponding input split;

AN_{hadoop}: number of map tasks executed by a node in native Hadoop on average;

$AN_{improved}$: number of map tasks executed by a node in improved Hadoop on average;

At_{hadoop}: time needed by the execution of a map task in native Hadoop on average;

$At_{improved}$: time needed by the execution of a map task in improved Hadoop on average;

t_{hadoop}: time needed by a map task with no local input data in native Hadoop;

$t_{improved}$: time needed by a map task with no local input data in improved Hadoop;

T_{hadoop}: time needed by a job in native Hadoop;

$T_{improved}$: time needed by a job in improved Hadoop;

The improvement of a map task with no local input data can be represented as:

$$\Delta t = t_{hadoop} - t_{improved} \tag{1}$$

$$\Delta t = \left(\frac{S_{buf}}{V_{tran}} + \frac{S_{buf}}{V_{map}}\right) * \frac{S_{map}}{S_{buf}} - \left(\frac{S_{buf}}{V_{tran}} + \frac{S_{buf}}{V'_{tran}} + \left(\frac{S_{map} - S_{buf}}{\min\left(V'_{map}, V_{tran}\right)}\right)\right)$$

To implement the data prefetching mechanism proposed in this paper, only two S_{buf} extra memories and a data prefetching thread are needed. Because these overheads are little, the V'_{map} is equal to V_{map} approximately. Thus Δt can be simplified as:

$$\Delta t \approx \left(S_{map} - S_{buf}\right) * \left(\frac{1}{V_{tran}} + \frac{1}{V_{map}} - \frac{1}{\min\left(V_{map}, V_{tran}\right)}\right)$$

$$\Delta t \approx \frac{S_{map} - S_{buf}}{\max\left(V_{map}, V_{tran}\right)} \tag{2}$$

The equation (2) proves that the data prefetching mechanism can overlap one of the data transmission process and the map process. The comparison between V_{map} and V_{tran} reflects the bottleneck of a map task. If V_{tran} is higher, the map process is the bottleneck and the data transmission process will be overlapped. Otherwise, the map process will be overlapped. In common, the data processing is slower than data transmission even in data-intensive applications, so this paper focuses on reducing the cost of data transmission. Now Δt is:

$$\Delta t \approx \frac{S_{map} - S_{buf}}{V_{tran}} \tag{3}$$

As shown in equation (3), a map task with bigger S_{map} gets bigger Δt. It means that the bigger the S_{map}, the more performance improvement a map task can achieve. But too big S_{map} will affect the concurrency of MapReduce, so in Hadoop the user is required to set this parameter depending on situations.

The improvement of a job is the aggregation improvement of map tasks executed by the node that returns result of the last map task. The determination of this node is affected by many factors including nodes' capability, distribution of input data, scheduling algorithm and backup tasks, which makes the analysis difficult.

The load balance of Hadoop is used to simplify this problem. In the default scheduling process of Hadoop, a node is assigned a new map task when it becomes capable, and the number of map tasks is about hundreds of times more than the number of nodes in MapReduce [1]. As a result, the distribution of tasks is almost in proportion to the capability of nodes and good load balance is achieved. So the node that returns result of the last map task can be replaced by an average node approximately. Then the improvement of a job can be represented as:

$$\Delta T = T_{hadoop} - T_{improved} \tag{4}$$

$$\Delta T \approx AN_{hadoop} * At_{hadoop} - AN_{improved} * At_{improved}$$

Obviously, AN_{hadoop} is equal to $AN_{improved}$ for the numbers of map tasks and nodes are same in native and improved Hadoop. Because the data prefetching mechanism affects the allocation of map tasks, the relationship of At_{hadoop} and $At_{improved}$ needs to be discussed in following cases:

Case 1: For tasks whose input data is local both in native and improved Hadoop,

$$At_{hadoop} = At_{improved} \tag{5}$$

Case 2: For tasks whose input data is local only in native Hadoop, the overhead caused by improved Hadoop is:

$$At_{improved} - At_{hadoop} \approx \frac{S_{buf}}{V_{tran}} \tag{6}$$

Case 3: For tasks whose input data is local only in improved Hadoop, the improvement brought by improved Hadoop is:

$$At_{hadoop} - At_{improved} = \frac{S_{map}}{V_{tran}} \tag{7}$$

Case 4: For tasks whose input data is not local both in native and improved Hadoop, the improvement brought by improved Hadoop is:

$$At_{hadoop} - At_{improved} = \Delta t \tag{8}$$

It can be observed that in case 1, case 3 and case 4, the performance of improved Hadoop is better than the native Hadoop. Even in case 2, the overhead caused by the improved Hadoop is little for the S_{buf} is only several KB.

According to the analysis above, the data prefetching mechanism is proved to hide data transmission time effectively both in single map task and job composed of multiple map tasks.

4. Experimental Evaluation

Experiments are carried out in heterogeneous and shared Hadoop clusters separately. The proposed data prefetching mechanism is implemented in Hadoop-1.0.4 with the default scheduler. With the virtualization solution KVM [12], eight virtual machines in Table 1 are created and connected by 100Mb Ethernet links. The wordcount and CCV evaluator [13] (Class-Center Vectors evaluator) are used as MapReduce applications. The wordcount is a simple application used for counting the number of words, and CCV evaluator calculates the class-center vectors used in classification algorithms. To evaluate the performance improvement of MapReduce in heterogeneous or shared environments, the replication factor of HDFS is set to 1.

Table 1. Configurations of nodes in Hadoop cluster

Node ID	N0	N1	N2	N3	N4	N5	N6	N7
Number of virtual processors	1	1	1	1	2	2	2	2
Memory sizes(MB)	256	512	1024	2048	256	512	1024	2048

4.1. Performance improvement in heterogeneous environment

As shown in the 3.2, the relationship between speed of data transmission and speed of map processing affects the performance of proposed mechanism. In experiments, delay is used in map processing and data transmission to simulate different conditions. Different jobs' configurations are given in Table 2.

The average time that map tasks in native Hadoop and improved Hadoop spend in getting corresponding input data is shown respectively is Figure 4.

It is observed that the data transmission times of all jobs are reduced effectively in improved Hadoop. The Job0 and Job1 spend almost the same data transmission time, which means that the data prefetching mechanism is effective for different map processing times. This conclusion is supported by the comparison of Job2 and other CCV evaluator jobs also.

Table 2. Configurations of jobs in experiments

Job ID	Job0	Job1	Job2	Job3	Job4	Job5
Application	word-count	word-count	CCV evaluator	CCV evaluator	CCV evaluator	CCV evaluator
Delay of map processing(ms)	10	20	0	1	1	1
Delay of 4KB Data transmission(ms)	1	1	1	1	2	3
Total input size of job(MB)	2078.72	2078.72	1957.12	1957.12	1957.12	1957.12
Input split size of map(MB)	64	64	61.16	61.16	61.16	61.16
Number of reduce tasks	1	1	1	1	1	1

With varied data transmission delays, the data transmission times of Job3, Job4 and Job5 in native Hadoop are quite different. But in improved Hadoop, the differences of data transmission times are overlapped by data prefetching, which makes these times are not affected by real times spent for data transmission. The experiment results show that the maximum reduction of data transmission time is achieved in Job5, which is up to 94%. That means data prefetching are especially effective for the poor network connections.

In a word, the proposed data prefetching mechanism can hide almost all data transmission time in different conditions.

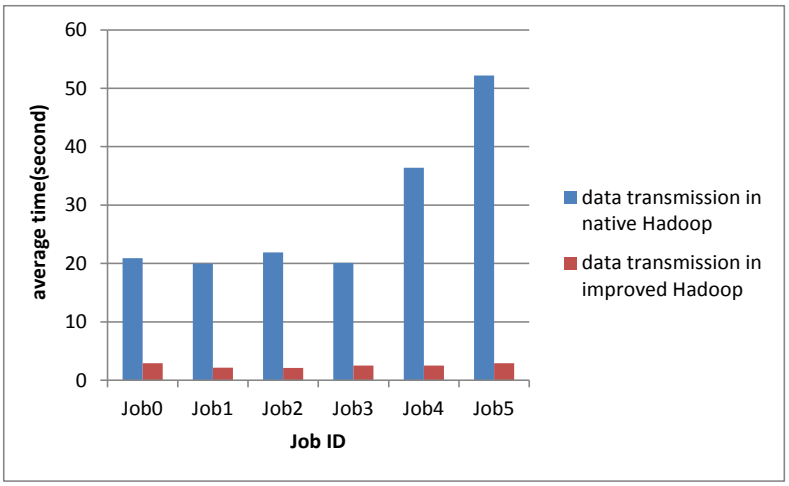

Figure 4. Time of data transmission

To inspect effect of total input data size and input split size of a map task on the proposed data prefetching mechanism, these two parameters of Job 2 are changed separately in Figure 5 and Figure 6.

In Figure 5, the execution time of jobs is reduced by 15% on average when the total input size is equal or bigger 2GB. When the total input size is 1GB, the number of map tasks is about 16 and each node only needs to execute about 2 map tasks, so the execution of job is improved a little.

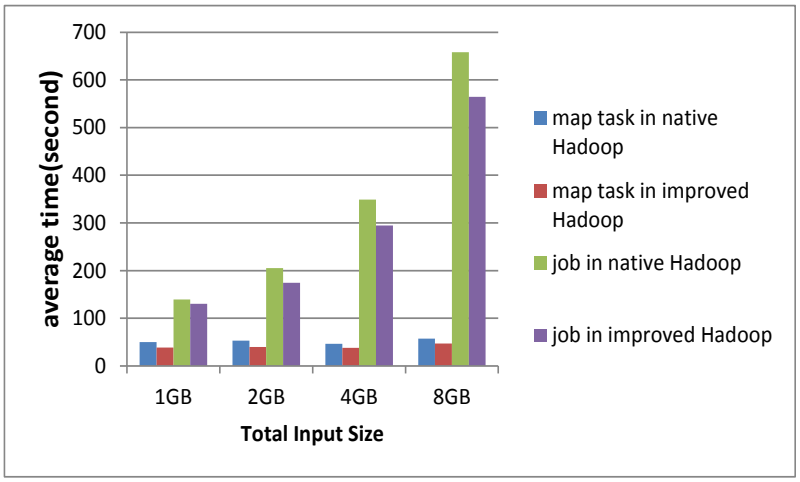

Figure 5. Performance with different total input size

Figure 6 shows that bigger input split size Smap brings more performance improvement for map tasks, which accords with the theoretical analysis in 3.2. It is observed that 25% performance improvement of map tasks is achieved when the input split size is 64MB.

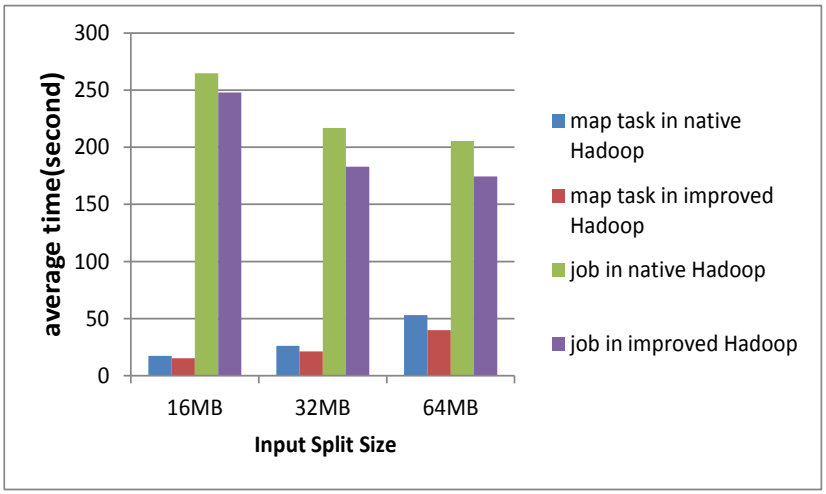

Figure 6. Performance with different input split size

4.2. Performance improvement in shared environment

In shared environment, a Hadoop cluster is used by many users, so nodes one user can use are often limited to part of nodes in cluster. To simulate a shared Hadoop cluster, the Job2 in Section 4.1 is executed by 2 nodes, 4 nodes and 8 nodes separately, and the results are given in Figure 7. It is observed that the execution time of job achieves 15.5% improvement when two nodes are used and 15.1% improvement when eight nodes are used. So with the proposed data prefetching mechanism, clusters shared by different numbers of users can benefit evenly.

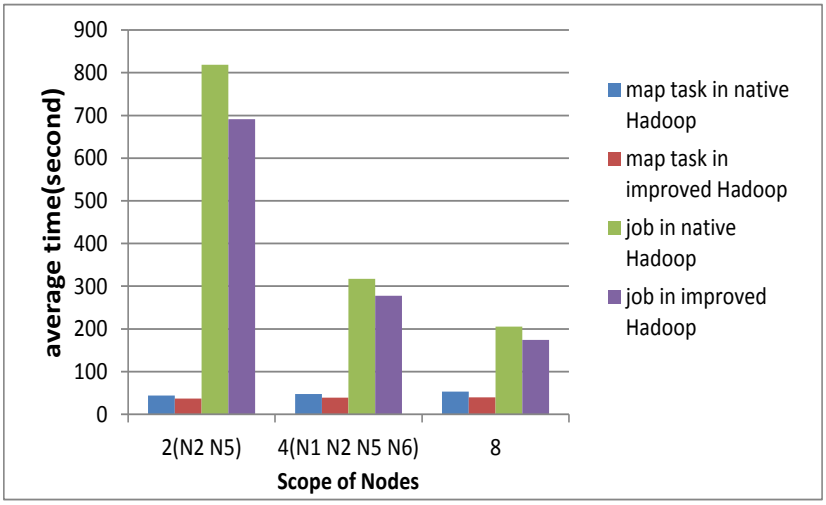

Figure 1. Performance with different scope of nodes

5. Related Work

To alleviate the performance degradation caused by data transmission, some related work is done. Data Prefetching is an effective approach to diminishing the data transmission overhead. To avoid directly modifying the native Hadoop, a bi-directional processing approach is proposed in HPMR [14]: computing fetches and processes data from the

beginning of the input split data while the prefetching fetches data from the end of the input split data. Obviously, the computing has to fetch data by itself before meeting the data fetched by the prefetching, which discounts the benefits of data prefetching. While the proposal in this paper fetches data from the beginning of input data to reduce the overhead of data transmission at the maximum.

Some researchers focus on optimizing task scheduling algorithms or data replication policies to improve data locality in MapReduce [8]. These proposals only improve the probability of data locality in MapReduce and may increase the complexity of achieving load balance. The LATE scheduling algorithm is proposed for MapReduce in heterogeneous environments [9]. M. Zaharia *et al.*, have proposed a delay scheduling algorithm, which addresses the conflict between locality and fairness in shared MapReduce cluster [15]. In MTSD [16], computing nodes are classified by computing capability and a modified task scheduling algorithm is studied. X. Zhang et al. have studied scheduling with consideration about data locality in homogeneous cluster [17]. DARE is a distributed adaptive data replication algorithm that is sensitive to the heterogeneity of computing nodes, and the more powerful nodes get more data replications [18].

6. Conclusions and Future Work

This paper aims at improving the performance of MapReduce in heterogeneous or shard environments degraded by poor data locality. By analyzing the process of map tasks in Hadoop, the serial execution of data transmission and data processing is discovered to cause overhead when the input data is not local. The proposed data prefetching mechanism overlap data transmission with data processing, thus the input data of map tasks can be prefetched to nodes where map tasks are executed. Through theoretical analysis of single map task and job composed of multiple map tasks, the proposed data prefetching mechanism is proved effective in overlapping data transmission process. Experiments are carried out both in heterogeneous and shared environments. Parameters including map processing time, data transmission time, total input data size and input split size of map tasks in experimental applications are also varied to simulate different conditions. The experiment results show that up to 94% data transmission time is reduced and up to 15% performance improvement in jobs' execution is achieved with the proposed mechanism.

The future work will focus on applying similar prefetching mechanisms to other phases in MapReduce, and researching on predicting the execution nodes of tasks in cluster computing to improve performance further. Coordinating the scheduling algorithm with data prefetching mechanism is also an interest topic to be discussed.

Acknowledgements

The work in this paper is supported by the National Natural Science Foundation of China (Grant No. 61212005) and the Fundamental Research Funds for the Central Universities of China (Grant No. 65012101).

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", Communications of the ACM, vol. 51, no. 1, **(2008**).
[2] D. Jiang, B. Chin, L. Shi and S. Wu, "The performance of MapReduce: an in-depth study", Proceedings of the VLDB Endowment. vol. 3, no. 1, **(2010)**.
[3] J. Dean and S. Ghemawat, "Mapreduce: a flexible data processing tool", Communications of the ACM, vol. 53, no. 1, **(2010)**.

[4] T. White, "Hadoop: The Definitive Guide (3rd Edition)", O'Reilly Media/Yahoo Press, **(2012)**.
[5] Hadoop Wiki Website, Apache, http://wiki.apache.org/hadoop/PoweredBy.
[6] S. Papadimitriou and J. Sun, "DisCo: Distributed Co-clustering with Map-Reduce: A Case Study towards Petabyte-Scale End-to-End Mining", Eighth IEEE International Conference on Data Mining, **(2008)** December 15-19: Pisa, Italy.
[7] R. Taylor, "An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics", BMC bioinformatics, 11(Suppl. 12):S1, **(2010)**.
[8] S. Khalil, S. A. Salem, S. Nassar and E. M. Saad, "Mapreduce Performance in Heterogeneous Environments: A Review", International Journal of Scientific & Engineering Research, vol. 4, no. 4, **(2013)**.
[9] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares and X. Qin, "Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters", IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), **(2010)** April 19-23: Arlanta, USA.
[10] H. Jin, X. Yang, X. Sun and I. Raicu, "ADAPT: Availability-aware MapReduce Data Placement for Non-Dedicated Distributed Computing", IEEE 32nd International Conference on Distributed Computing Systems (ICDCS), **(2012)** June 18-21: Macau, China.
[11] K. Shvachko, H. Kuang, S. Radia and R. Chansler, "The Hadoop Distributed File System", IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), **(2010)** May 3-7: Incline Village, USA.
[12] KVM Wiki Website, http://www.linux-kvm.org/page/Main_Page.
[13] X. Zhang, "Using Class-Center Vectors to Build Support Vector Machines", IEEE Signal Processing Society Workshop Neural Networks for Signal Processing IX, **(1999)** August 23-25: Madison, USA.
[14] S. Seo, I. Jang, K. Woo, I. Kim, J. S. Kim and S. Maeng, "HPMR: Prefetching and Pre-shuffling in Shared MapReduce Computation Environment", IEEE International Conference on Cluster Computing and Workshops, **(2009)** August 31-September 4: New Orleans, USA.
[15] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker and I. Stoica, "Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling", Proceedings of the 5th European conference on Computer systems, **(2010)** April 13-16: Paris, France.
[16] Z. Tang, J. Q. Zhou, K. L. Li and R. X. Li, "MTSD: A task scheduling algorithm for MapReduce base on deadline constraints", IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW), **(2012)** May 21-25: Shanghai, China.
[17] X. Zhang, Z. Zhong, S. Feng and B. Tu, "Improving Data Locality of MapReduce by Scheduling in Homogeneous Computing Environments", IEEE 9th International Symposium on Parallel and Distributed Processing with Applications (ISPA), **(2011)** May 26-28: Busan, Korea.
[18] C. Abad, Y. Lu and R. Campbell, "DARE: Adaptive Data Replication for Efficient Cluster Scheduling", IEEE International Conference on Cluster Computing (CLUSTER), **(2011)** September 26-30: Austin, USA.

