2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing

The Dynamically Efficient Mechanism of HDFS Data Prefetching

Shaochun Wu, Guobing Zou, Honghao Zhu,Xiang Shuai,Liang Chen,Bofeng Zhang
School of Computer Engineering and Science, Shanghai University
Shanghai, China
scwu@shu.edu.cn, guobingzou@gmail.com, 15170015668@163.com

***Abstract*—In recent years, along with cloud computing developing as a widely used computing paradigm, Hadoop Distributed File System (HDFS) has become one of the mandatory techniques, which has many important features, such as master and slave construction of HDFS, direct client accessing, and multi-duplicate of each data block. All of these make HDFS data prefetching much harder than the traditional data acquisition approaches. Moreover, the basic problems of HDFS data prefetching mainly include what kind of data to prefetch, where to prefetch data, how many data to prefetch, and the balance of prefetching data services and normal data access conflicts. Under above analysis, this paper tries to solve these problems and propose the mechanism of the two-layer HDFS data prefetching. The experimental results show that the Hadoop platform which offers data prefetching mechanism can improve 60% of whole performance on data prefetching.**

Keywords—*Cloud computing, HDFS, Data prefetching, Metadata, Data block*

I. INTRODUCTION

The Hadoop Distributed File System (HDFS) is a representative of Internet file system, which has been widely used in Web applications. For most of the Internet applications, data access delay which caused by access data from the file system has become the main cost of processing user requests. This cost is even more obvious in the Internet applications based on Hadoop. Due to the HDFS data access mechanism, data access delay has encountered serious drawbacks when accessing data from HDFS, especially access to a large number of small files. In general, there are three kinds of impacts on the performance of reading file from the HDFS. These three aspects consist of the delay caused by each file reading metadata from the metadata server, the I/O delay for reading a file from the disk, and the network delay when transferring files to an end client.

Data prefetching is a simple and effective technique to reduce data access delay. In order to hide visible I/O latency and shorten the response time for a user request, it is effective to prefetch the data to a local disk based on the limit of data access before a user submits a data access request. On the other hand, the data that Internet applications need to access has so many limitations. Data prefetching techniques can solve the problem of HDFS data access latency and effectively improve the performance of HDFS. However, it cannot be directly applied to the HDFS for data access, due to some personalized features of HDFS as we have mentioned before, such as the master and slave construction of HDFS, direct client accessing, and multi-duplicate of each data block. All of these make HDFS data prefetching much harder than the traditional.

II. RELATED WORK

The Lin Lin et al.[6] proposed a method for the optimization of HDFS to improve the performance of Hadoop. It analyzed the reason of HDFS poor performance is the hypothesis of HDFS portability. As we know, in order to enhance the portability, HDFS used Java to implement the platform, which is a potential reason why the HDFS is inefficient when accessing data for users. While Java enhanced Hadoop portability, it shields the underlying file system and makes it cannot use some of the underlying APIs to optimize read-write and data storage. First, a large number of concurrent read-writes increase the time of random seeking in the environment of shared cluster, which reduces the efficiency of read-write operation. Then, concurrent write will increase the disk fragmentation as well as the cost of read. As far as this problem, some researchers have improved task tracker thread model, i.e., the Hadoop of nowadays is the model of a thread corresponding to a client. The threading model can deal with the client-side communication and store data at the same time. One kind of improvement of this thread model is divided thread into two groups, one is to deal with the client communication, and the others are to store data. Using the thread model, we can solve the problem effectively. For improving the efficiency of threading model, Lin Lin et al. [6] can enhance the performance of Hadoop. The goal of this paper is to enhance the performance of the HDFS data read which can be provided by a two-layer data access echanism.

Yong Chen et al.[7] proposed a strategy to optimize the performance of Hadoop by data prefetching technique. The strategy included two-layer prefetching: within block prefetching and interblock prefetching. The functionality of within block prefetching is to optimize the mode of internal data processing. The two-way processing strategy can improve the efficiency of data access in Hadoop by calculating the data and then prefetching the data. To apply this strategy for accessing data from Hadoop, two issues need to be solved. First, it is necessary to calculate and control the synchronization of data prefetching. To solve this issue, this paper uses the progress bar to monitor status. Thus, when the synchronization process is broken, a signal is called to deal with the problem. Second, it must determine the appropriate rate of data prefetching.

III. TWO-LAYER MECHANISM OF HDFS DATA PREFETCHING

A. Metadata prefetching

978-0-7695-5046-6/13 $26.00 © 2013 IEEE
DOI 10.1109/GreenCom-iThings-CPSCom.2013.413

M. Stonebraker et al.[5] proposed that the correlation information can be used to help prefetch file data, especially improve the prefetching accuracy of the HDFS file metadata. In this paper, the correlation file information provide a foundation of the metadata prefetching algorithm to reduce the cost of HDFS file data prefetching and effectively alleviate the performance bottleneck of the metadata server.

In a typical storage system, because there are hundreds of metadata needs to be updated at the same time, Shivnath Babu et al. [1] proposed metadata access operations account for most of all I/O operations. That is, the metadata server is a performance bottleneck of a distributed storage system. We often solve the performance bottlenecks of metadata server from two aspects. On one hand, ultiple metadata servers can be used to coordinate metadata requests for the system load balance. On the other hand, the distributed system can reduce metadata operation costs by improving the hit ratio of storage system.

In the HDFS, D. Jiang, B. C et al.[2] proposed there is 50% of I/O operation associated with metadata access, and Yu SZ et al.[8] proposed typically the normal size of file metadata is less than 5% of file data. In particular, positive data is one of strategies by prefetching a large amount of data to improve system performance. Since the metadata is relatively small, HDFS can improve performance through the strategy of positive metadata prefetching, when the incorrect prefetching ratio is low. However, if prefetching is inaccurate, then the error prefetching can quickly eliminate the performance improvement which is caused by active prefetching. In order to alleviate this problem, we set the threshold of file relevancy, and remove the irrelevant files or the files that are below the threshold value from the list of related documents. Through reasonable setting of the relevancy threshold of files, we can acquire near-optimal performance under the prefetching circumstances where mistakes will happen as low as possible.

Metadata prefetching algorithm only needs to prefetch the metadata information of related file list to the NameNode cache. By adopting a positive data prefetching strategy, together with the analysis of document relativeness under a predefined relevance threshold, we can substantially improve data prefetching accuracy. The NameNode proposed by A. Abouzeid, K et al.[3] uses the LRU algorithm to replace the invalid metadata in cache.

B. The file data prefetching

a. The file data prefetching model

Peng Xia et al.[9] proposed Prefetching file can hidden I/O latency, and reduce the network transmission delay. In order to solve the four fundamental problems difficult in HDFS data prefetching, we propose a data prefetching model, as shown in the following Figure 1.

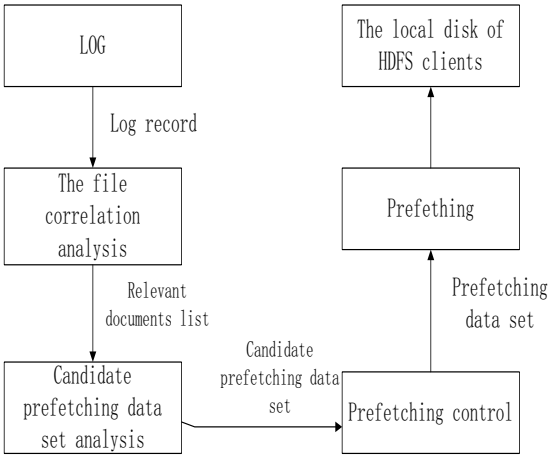

Figure 1 Prefetching data model

In Figure 1, document relevance analysis by mining related file list from the log is to predict the related files which it is used as references for data prefetching. Obviously, the more correct the document relevant analysis is, the more useful prediction will be for data prefetching. Therefore, to find an algorithm that can really reflect the degree of document relevance. As is known to us, data prefetching technique is not a kind of highly efficient optimization technique. Therefore, how to coordinate data prefetching and normal data access services is important. A. Pavlo et al.[4] proposed Prefetching control is mainly to control the balance between data prefetching services and normal data access service. At the same time, according to the usage of current network resources and the load of server to control the amount of prefetching and its associated threshold, we can avoid bad influence on other network applications. By controling data prefetching coordination, candidate prefetching data set can help get the data set which we really need. Prefetching data set through the prefetching module read the data set into local disk of HDFS client-side. Before the client-side needs to access data from the HDFS, it should read the data from local disk primarily. If the data set the client-side want is not prefetched and stored in local disk, then it accesses the data by normal HDFS data access processes. When reading a request is missing in the metadata cache, it will trigger a metadata prefetching and file data prefetching.

b. The dynamic adjustment mechanism of data prefetching quantity

In addition to prefetching data, control the quantity of data prefetching should also be taken into account in order to keep sufficient accuracy in the process of data prefetching. Existing data prefetching mechanism mainly adopts two strategies, including conservative prefetching and active prefetching. Conservative prefetching strategy reduces unnecessary prefetching as much as possible, in order to minimize the network load caused by prefetching process. However, it can also incur low hit rate of data prefetching, such that data prefetching cannot achieve the desired performance. On the contrary, active prefetching strategy is to prefetch data as much as possible, which might cause excessive prefetching

and potentially increase network transmission load, but can improve the hit rate of prefetching data

Conservative prefetching or active prefetching both have their advantages and disadvantages. When analyzing these two kinds of prefetching strategies, the prefetching process to adjust the prefetching quantity mainly considers the following two factors:

1) Network condition. Network condition is an exterior factor that affects the performance of data prefetching. When network bandwidth is wider and latency is lower, the prefetching process should adopt a positive prefetching strategy, and the quantity of data prefetching can be larger than before. Conversely, conservative prefetching strategy should be adopted and the quantity of data prefetching must be small to adapt to the network load.

2) The client-side characteristics of read requests. The read request characterized of client-side is an internal factor to influence the data prefetching. In the period of time when read requests are intensive, the local hit rate can be improved more effectively by a larger prefetching quantity. Conversely, in order to reduce the prefetching effects on normal data access service, it should be appropriate to reduce data prefetching quantity.

In this work, we use a metric, called the change of network transmission rate, to measure the dynamic changes in a specified network condition. The Network transmission rate (Network Speed, NS) can be obtained through the measurement of the prefetching process. If T period of time is spent in prefetching n number of data blocks, then NS can be calculated as NS = n/T. Suppose that two times in NS measurements are denoted as NS0 and NS1, respectively. Similarly the ith time is denoted as NSi. Then the change proportion of the network transmission rate, denoted as ΔNS, can be computed according to the following formula.

$$\Delta NS=\begin{cases}(\mathrm{NS1}-\mathrm{NS0})/\mathrm{NSi}, & \text{if NS1<NS0, NSi=NS1}\\ \mathrm{NSi}=\mathrm{NS0}, & \text{otherwise}\end{cases} \tag{1}$$

We use the alteration ratio of local failure hit to measure the degree of reading requests change of client-side. By the number of failure data block in each time, local hit failure MC (Missing Count) can be measured. Suppose that two invalid data blocks is MC0 and MC1 in two time slice, similarly the ith data blocks is MCi. then we have

$$\Delta MC=\begin{cases}(\mathrm{MC0}-\mathrm{MC1})/\mathrm{MCi}, & \text{if if MC1<MC0, MCi=MC1}\\ \mathrm{MCi}=\mathrm{MC0}, & \text{otherwise}\end{cases} \tag{2}$$

In order to make the prefetching can satisfy the read request, we maintain HDFS BS (Block Size) as the minimum value of prefetching quantity. For HDFS, the block size can be set through the configuration file. Therefore, it can be obtained by reading configuration file during the prefetching process. Within each time slice, we make the increment of NC and MC and BS as the prefetching quantity unit to adjust the quantity of data prefetching. When the prefetching quantity is greater than a certain value, the local hit rate will be markedly reduced. Due to the dramatic changes of the NS and MC, it has to avoid the quantity of data prefetching too big or too small. By using a bounded increment adjustment factor, we can apply its changes with the index to control the changes of data prefetching quantity. Let T as average increment, as shown in formula (3), then the adjustment factor of prefetching quantity increment is calculated by the formula (4). Assume that last prefetching quantity is Pn-1, then during the current time slice the prefetching quantity Pn can be calculated by the following formula (5):

$$T=(\Delta NC+\Delta M)/2 \tag{3}$$

$$\eta=(-1)^t\times[10-20e^{T/8}/(e^{T/4}+1)],\quad \text{if } T>0,\ \text{then } t=0;\ \text{else } t=1 \tag{4}$$

$$p_n=\eta\times BS+p_{n-1},\ \text{if } p_n<BS,\text{then} p_n=BS \tag{5}$$

c. The collaboration strategy of data prefetching

Different HDFS client-side's concurrent data access requests and data prefetching requests will lead to the consumption of HDFS computing resources and network bandwidth. In order to improve the comprehensive performance of the whole system, this paper uses a global collaboration strategy to decide what kind of data eventually need to be prefetched.

Both two services will consume HDFS computing resources and network bandwidth, thus it will cause resource competition. Relative to the data prefetching service, data access service should have higher priority, because the quality of service has more important influence on the performance of the entire application. In order to reflect the different priority between data prefetching service and data access service, we introduce the priority weights PWa and PWp, which represent the priority of data access service and data prefetching service, respectively. These two priority weights conform to the following conditions: PWa > PWp , and PWa + PWp =1. In order to elaborate the collaboration strategy, we formulate the following definitions.

Definition 1: PI (Penalty Index）. Penalty index, denoted as PIj, represents the delay time that another node accesses the data when the client-side j prefetching the data, under the condition of no comparison with data prefetching. It is calculated as below.

$$PI_j=\frac{\sum_{i\in S_{AL(j)}} is\,\Pr efetch(i)\times S_i}{PW_P\times[\sum_{k\in S_{active},k\neq j}(\sum_{i\in S_{AL(j)}} is\,\Pr efetch(i)\times S_i)]+PW_a\times\sum_{k\in S_{active}} ts_k} \tag{6}$$

ts(k) is the total number of dimensions of the data that client-side k would like to prefetching, but not including the prefetching data

$$S_{active}=\{i\mid \text{The data client is requesting HDFS service}\} \tag{7}$$

$$is\,\Pr efetch(i)=\begin{cases}1, & \text{If data i is prefetched}\\ 0, & \text{If data i do not be prefetched}\end{cases} \tag{8}$$

Where, $S_{AL(j)}$ is the candidate prefetching data set of client-side.

Definiton 2: BI (Benefit Index). Benefit index is defined as the increase proportion of prefetching performance when compared with no data prefetching. It is measured by

$BI_i = \frac{T_{prefetch}}{T}$, where, $T_{prefetch}$ is the time to access prefetching data, while T is the time to access the data without data prefetching.

Based on above analysis, the collaboration strategy of data prefetching is to balance the effectiveness and the delay time. Thus, the collaboration strategy can be formulated as the following optimization problem:

$$\max\{\sum_{i\in S_{pre}} is\,\Pr efetch(i)\times BI_i - \sum_{s\in S_{active}} PI_k\} \quad (9)$$

Where, S_{pre} is the data set that needs to be prefetched. The solution space of the coordination optimization problem is the value of isPrefetch(i) that makes the expression achieve maximal. Through the collaboration strategy, we can obtain set S which involves final items of prefetching data.It can be expressed as:

$$S=\{i|is\,\Pr efetch(i)=1\},\ \text{and}\ S\subseteq S_{pre} \quad (10)$$

C. Prefetching process

The main process of data prefetching from HDFS consisting of server correlated steps is as follows.

Step1: When a user submits a read request, we then search the metadata files in NameNode cache primarily. If metadata files exisit, we directly jump to Step 6, otherwise jump to the following Step 2.

Step2: Client user sends the request to the NameNode.

Step3: In the NameNode, data prefetching module sends the file name which the user uses to file correlation analysis module.

Step4: Document correlation analysis module according to the file name analyzes relevant documents, and then returns a relevant file list for metadata prefetching module and file data prefetching module.

Step5: Metadata prefetching module searches metadata of related files primarily, and then returns them to the client as well as stores into NameNode cache.

Step6: HDFS client testes whether the required data block is already in the local disk. If it is ready to use, then it reads the data block into HDFS client cache, the process ends; otherwise, it triggers the file data prefetching module.

Step7: File data prefetching module reads the data that needs to be prefetched into the local disk of HDFS client.

D. The monolithic construction of prefetching data

In the Hadoop platform, we mainly analyze the relevance of documents to guide data prefetching, and then optimize the performance of Hadoop distributed architecture. The structure of data prefetching in Hadoop is shown in Figure 2

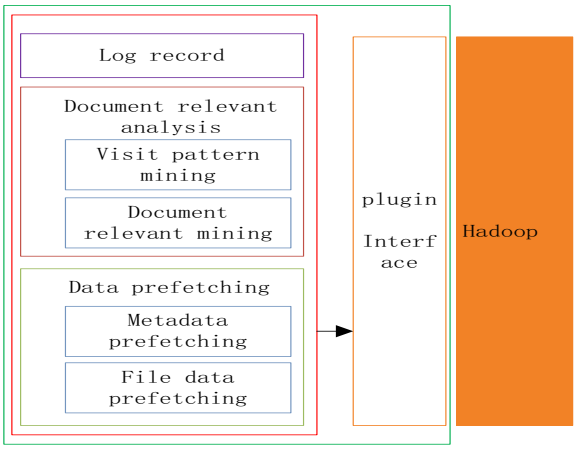

Figure 2 The structure of data prefetching in Hadoop

The above framework of the data prefetching in Hadoop consists of three key components, including log record module, document relevance analysis module, and data prefetching module. The log records and document relevance analysis modules do not interact with native Hadoop , while only data prefetching module through the plug-in interface needs to interact with the native Hadoop. Log record module records the process data that the users access in the log file, provided for the use of document relevance analysis module. Document relevance analysis module mines the user data access patterns from the log data by sequence mining algorithms, and then analyzes the relevance of files through the data access mode. The goal of document relevance analysis is to improve the accuracy of data prefetching. Intelligent analysis module aims at avoiding unnecessary data prefetching. Data prefetching module is to prefetch data, which consists of metadata prefetching module and file data prefetching module. According to the relevant of files, metadata prefetching prefetches the relevant list, which can be shared by multiple users in the cache. The file data prefetching prefetches the file that is located in the file relevant list to client's local disk, which can reduce data delay and improve the performance of data prefetching in Hadoop.

IV. Experimental Evaluation

A. Experimental Setup

The cluster of running evaluation experiment has seven nodes. One of the nodes is NameNode, the other six nodes are DataNode. Six DataNode are divided into two groups. The machine between two groups is connected with routers, which is used to imitate the cluster. Nodes in each group are connected by switchboard, which means the node belongs to the same rack.

Each node installs Ubuntu Linux operating system (the kernel version is 2.6.30), Hadoop 0.20.203，JDK 1.6.0_27. Hadoop NameNode and JobTracker run on the same node, and the other nodes run the DataNode and TaskTracker at the same time. HDFS block size is set to 64 MB and each block has preserved 3 copies.

In order to facilitate the comparison of experimental results, Hadoop platform which adds the function of data prefetching is called SHU-Hadoop, while original Hadoop platform is called Native Hadoop. We use two experiments to evaluate the performance of SHU-Hadoop. The first is the Wordcount workflow where Hadoop platform is integrated as one component. Wordcount is a chief standard program that Yahoo usually uses to evaluate the Hadoop performance. This experiment is mainly used to test the whole performance of SHU-Hadoop. The second experiment downloaded multiple files from multiple HDFS clients. The experiment mainly tests the whole throughput of SHU-Hadoop platform, the utilization rate of NameNode CPU, the network traffic of communication, and other performance parameters.

B. Experimental Results

In order to evaluate SHU-Hadoop's performance, we use the wordcount benchmark test program which has plugged in Hadoop platform. However, in order to imitate the complex task, we have modified original wordcount program.

We utilize the sleep function when counting each word in order to achieve 1 ms latency. At the same time, in order to test the influence of multiple factors on the whole performance, five experiment sets have been adopted for experimental evaluation, and each experiment set runs in different nodes and Map tasks number.

In addition, each experiment in a given experiment set has been done more than five times at least. The configuration of each experiment set is shown in Table 1:

	1	2	3	4	5
workflow	wordcount	wordcount	wordcount	wordcount	wordcount
Node	2	4	5	6	6
number of Map task	6	6	6	10	6
number of reduce task	1	1	1	1	1
size of input file	4.4GB	4.4GB	4.4GB	4.4GB	4.4GB

Table 1 The configuration of experimental dataset

C. Experimental Analysis

To evaluate the efficiency of our modified HDFS with the native one, we compare the performance between SHU-Hadoop and Native Hadoop. We measure the time that spends in the experimental platform which completes the relevant workflow. The time that SHU-Hadoop and Native Hadoop process relevant workflow under each experiment dataset is shown in Figure 3.

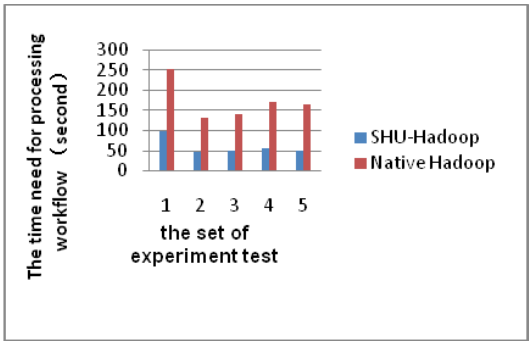

Figure 3 Chart of process workflow time comparisons

From the experimental results, we can conclude that from Figure 5, for all of the experimental datasets, SHU-Hadoop performance is better than Native Hadoop performance. Compared with Native, SHU-Hadoop performance has improved 60% or more, especially the best case is 73.2%. Note that, the result of the experiment dataset 1 shows that the performance is worst among all of the experimental datasets. Firstly, it is because the environment of experimental dataset 1 has the smallest proportion between node number and Map tasks number. Thus, it will increase the dispatch cost.In other word, when executing the experiment in experimental dataset 1, we need to spend additional time to allocate task. Secondly, it is difficult to make use of the data block location information when assigning tasks. Additionally, because experimental dataset 1 is configured with the minimum number of nodes to perform a task, it is highly possible that data block of task is located in other nodes. Consequently, despite the experiment dataset 1 and dataset 2 using the same input file, the performance of experiment dataset 1 is worse than that of experiment dataset 2.

Finally, among all of the deployed experiments, the experiment dataset 5 shows the best performance. Compared with experiment dataset 4, although they have the same number of nodes to handle workflow and the number of experiment dataset 4 Map task is more than experiment dataset 5, the performance of experiment dataset 5 is better than that of experiment dataset 4. The reason is that MapReduce has a moment for reshuffling between Map tasks and Reduce tasks, so it needs to cost some system resources. Because the experiment dataset 4 has more Map tasks and needs to cost more system resources in shuffle phases, the performance of experiment dataset 4 is inferior to experiment dataset 5.

V. CONCLUSION

This paper proposes the data prefetching mechanism which uses sequence mining to forecast the file relevance, in order to improve the accuracy of the document relevance prediction. Moreover, during the prefetching time, it needs to control the prefetching data in order to increase the hit rate when prefetching data to be accessed in the future, and divides the prefetching data with different slackness into different queues, in order to reduce the prefetching influence on network bandwidth. We can see from the experimental results, the

whole performance of SHU-Hadoop is greatly enhanced. When reading the data concurrently, the total throughput of reading is increased 146.15% on average, and the increase of network traffic is only 66.48% on average. Relative to the improvement of throughput, the increase of network traffic is acceptable. But when there is a great number of client and low accuracy rate of documents relevant forecast, the improvement of read throughput is only 60.07%.

The master/slave structure of HDFS, direct client access, multiple copies of each block, all of this make HDFS data prefetching more complex than traditional data prefetching. So the data prefetching mechanism in Hadoop platform has many problems need to be solved. In this field has the following several aspects worthy of further expansion and deepening:

Firstly, HDFS is rack awareness, when select a copy to read, it's always trying to choose the copy which most close to the request node. This strategy will lead to "hot" data nodes are frequently access, which influence the load balancing of Hadoop platform. Therefore, design a duplicate selection algorithm to solve the overheating copies selected problems of current Hadoop platform, improve the efficiency of the Hadoop platform data prefetching and also can ensure load balancing has important research significance.

Secondly, Data prefetching is closely related to the scheduling algorithm of Hadoop, data prefetching mechanism proposed in this paper for some scheduling algorithm has good performance, while for other scheduling algorithm the performance is not very good. Therefore,It has important research significance that adopt dynamic data prefetching mechanism according to the different scheduling algorithm for ensure the performance of data prefetching.

ACKNOWLEDGMENT *(Heading 5)*

We thank all of the anonymous reviewers for their insightful suggestions and comments that will significantly improve the quality of our manuscript. This work was financially supported by the Ocean Public Welfare Project of The Ministry of Science and Technology under Grant No.201105033, the National Natural Science Foundation of China. (No.40976108),

REFERENCES

[1] Shivnath Babu. Towards automatic optimization of MapReduce programs. In Proceedings of the SoCC. 2010.

[2] D. Jiang, B. C. Ooi, L. Shi, S. Wu: The Performance of MapReduce: An In-depth Study. In Proceedings of the Very Large Databased (VLDB), 2010.

[3] A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi,A. Silberschatz, and A. Rasin. Hadoopdb: an architectural hybrid of mapreduce and dbms technologies for analytical workloads. In Proceedings of the VLDB, 2009.

[4] A. Pavlo, E. Paulson, A. Rasin, et al. A comparison of approaches to large-scale data analysis. In Proceedings of the SIGMOD, ACM , 2009.

[5] M. Stonebraker, D. Abadi, D. J. DeWitt, et al. Mapreduce and parallel dbmss: friends or foes. Communications of the ACM, 53(1):64-71, 2010.

[6] Lin Lin, Xuemin,Hong Jiang,Yifeng Zhu. AMP: An affinity based metadta prefetching scheme in large-scale distributed storage systems. In Proceedings of the CCGRID, 2008.

[7] Yong Chen, Huaiyu Zhu, Xian-He Sun. An adaptive data prefetcher for high-performance processors. In Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, 2010.

[8] Yu SZ., Kobayashi H. A new prefetch cache scheme. In Proceedings of the IEEE Global Telecommunication Conference, 2002.

[9] Peng Xia, Dan Feng, Hong Jiang, et al. FARMER: a novel approach to file access correlation mining and evaluation reference model for optimizing peta-scale file system performance. In Proceedings of the the 17th ACM Symposium on High Performance Distributed Computing (HPDC), 2008.