Vol.2010-DBS-150 No.7
Vol.2010-IFAT-99 No.7
2010/8/4

Comparing Hadoop and Fat-Btree based access method for Small File I/O Applications

Min LUO[†1] and Haruo YOKOTA[†1]

Hadoop has been widely used in various clusters to build scalable and high performance distributed file systems. However, Hadoop distributed file system (HDFS) is designed for large file management. In case of small files applications, those metadata requests will flood the network and consume most of the memory in Namenode thus sharply hinders its performance. Therefore, many web applications do not benefit from clusters with centered metanode, like Hadoop. In this paper, we compare our Fat-Btree based data access method, which excludes center node in clusters, with Hadoop. We show their different performance in different file I/O applications.

1. Introduction

Recently, the "cloud" has attracted considerable attentions in high performance and scalable distributed systems research. In these systems, a large number of servers are lined up and work in parallel. Many distributed execution frameworks have been proposed, in which, Map-Reduce in 2) is one of the most famous frameworks in these works. There are numerous academic and commercial implementations of Map-Reduce framework because it offers a simple, functional interface that transparently executes the computations with a good system scalability.

The most popular and public available Map-Reduce based project is Hadoop 9), an open source version under development by Apache Software Foundation. Its core components include an implementation of Map-Reduce, with a primary storage system called "Hadoop Distributed File System (HDFS)" to provide a conceptually simple programming model and abstract away any knowledge of where the data lives. Therefore, Hadoop has been widely used in both commercial and academic world, and shows high performance in many parallel processing tasks 10)–11).

On the other hand, many data access methods in parallel databases 4)–6), are other choices for data processing on the "cloud". Besides the contributions made by almost all the famous database vendors that are attracted in parallel databases over the past two decades, many ongoing academic projects are also engaged to provide better performance, scalability and failure tolerance parallel database systems 3), 12)–14). Although the parallel database and Map-Reduce based systems may seem to target different applications, it is in fact possible to write the parallel processing tasks for almost all the applications with Map-Reduce jobs or database queries with these two systems, individually 15). Therefore, it is meaningful to have some comparisons between these two systems for the users information.

In addition, in most recently, there is a new trends in the parallel processing system which tries to build a hybrid system based on Map-Reduce framework and Parallel DBMS to allow better code reusability, data independence and automatic query optimization 15). However, almost all the forerunners do not achieve the performance of parallel databases as well as the scalability, fault tolerance of MapReduce-based systems. One of the reasons is the different the storage layers, the HDFS layer or DBMS layer, they adopted. Because these storage layers have different performance in storing different size of data, it seems interesting and necessary to compare these two basic systems in different file I/O applications, and this comparison results may be referred in the design of future hybrid Map-Reduce/DBMS systems.

The purpose of this paper is to show the different I/O performance of Hadoop in the applications of different file sizes, especially for the small file I/O case. We compare the time consumed by Hadoop and a parallel DBMS in initializing and accessing all their data, individually. The overhead for accessing files in Hadoop is then discussed. In addition, a parallel database system based on Fat-Btree index 6) is introduced and used in this comparison, whose better scalability and availability than the Hadoop's are also shown through the experiments and discussion.

The rest of the paper is organized as follows: Firstly, we present background

†1 Department of Computer Science, Graduate School of Information Science and Engineering, Tokyo Institute of Technology

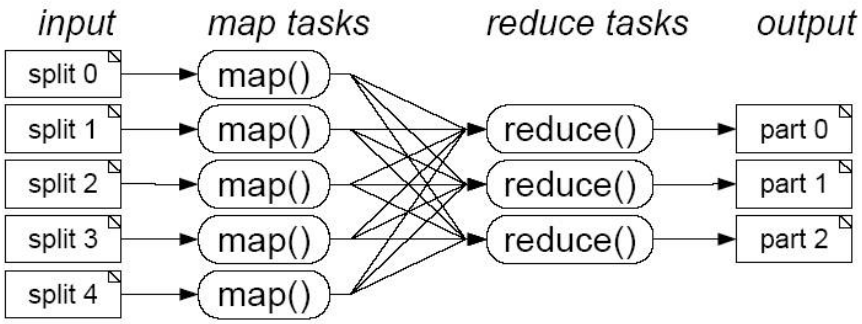

Fig. 1 Map-Reduce Model

work in Section 2. Then we outline the different maintain cost in Hadoop and parallel DBMS from architecture perspective in Section 3. In Section 4 we demonstrate and analyze the different performance in Hadoop and parallel DBMS in different file I/O applications through experimental results. Finally, we present related work in Section 5 and conclude our work in Section 6.

2. Background

2.1 Map-Reduce

Map-Reduce model (**Fig. 1**) consists two main functions, called Map and Reduce.

In the "Map" function, the master node reads a set of "records", chops it up into smaller intermediate records in a form of new (key, value) pairs, during this process, a "split" hash function partitions the records into R disjoint buckets. Each map bucket is written to the processing node's disk with these R output files. Since each map instance is assigned a distinct portion of input "records" by the scheduler, the total intermediate files created in this process is M*R, if there are M such distinct portions. Then, "Reduce" function transferred the intermediate files over network from the Map node's local disks to individual reducers, which is called "shuffle". Note that all the intermediate records with the same hash values are send into the same reducer and each reducer processes or combines the records into the final output file.

The advantage of the Map-Reduce is that the parallel and distributed processes are actuated automatically just by calling the MAP and Reduce operations provided within this model. In contract, parallel DBMSs may require their programmers to participate into this process, like providing the Map and Reduce processers with User Defined Functions and aggregation operations in DBMS.

2.2 Hadoop

Hadoop is an open source software for reliable, scalable, distributed computing which has two main components: a freely available implementation of Map-Reduce framework and a Hadoop Distributed File System (HDFS) 9).

HDFS is a distributed file system which is suitable for distributed processing on commodity hardware. It replicas the datasets on multiple nodes to make the data available even there is a failure of nodes. There are two kinds of nodes in HDFS: a metadata server called Namenode and a large number of data storage nodes called Datanode. The Namenode is in charge of all metadata and system actions data within the HDFS. The Datanode is in charge of all read/write and data replication requests according to the direction from Namenode. Because there is only one Namenode in Hadoop and it keeps all the metadata in main memory, it appears to be the bottleneck for handling metadata requests in the applications, especially for the applications on small files 16). In addition, this critical Namenode introduces an SPOF (Single Point of Failure) of the system, which is not easy to remove 25).

2.3 Fat-Btree

The Fat-Btree is a kind of parallel B-tree structure, which is proposed to provide dynamic data management, high throughput and efficient skew handling 6).

As shown in **Fig. 2**, the leaf pages of the parallel B-tree are distributed among the "Process Elements" (PEs). Each PE has a subtree of the whole B-tree containing the root node and intermediate index nodes between the root node and leaf nodes allocated to that PE. In the Fat-Btree structure, index nodes close to the root node have multiple copies but with a relatively low update frequency, on the other hand, leaf nodes have a relatively high update frequency but are not duplicated. Thus, nodes with higher update frequencies have lower synchronization overhead. Therefore, the maintain cost in Fat-Btree is much lower than the ordinary parallel Btree structure, such as Copy-Whole-Btree and Single-Index-

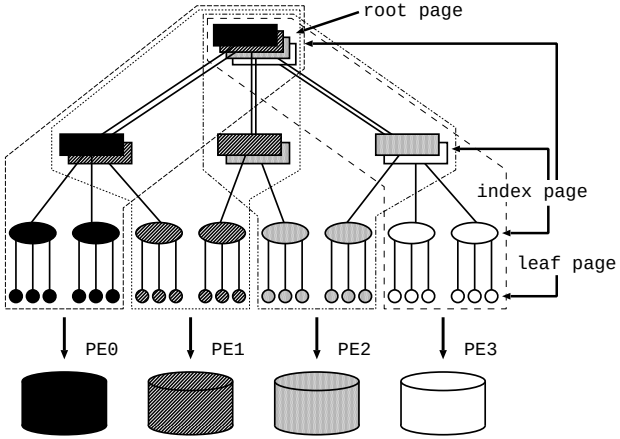

Fig. 2 A Fat-Btree

Btree in 6). We have proposed a parallel database system based on this Fat-Btree index in 20).

3. System Discussion

In this section, we discuss some of the system operation cost in Hadoop and Fat-Btree DBMS.

In Hadoop architecture, Map-Reduce model offers a simple, functional interface for distributed processing, while HDFS provides a reliable, shared virtual storage device.

HDFS stores the datasets and all the intermediate results which are generated during Map-Reduce processes across multiple nodes. In addition, HDFS cannot be directly mounted on an existing operating system, like Google File System (GFS), data needs to be loaded into HDFS before and after each execution in Hadoop. This can be very time consuming and we will examine this in the next section. Moreover, the HDFS store the files in each individual "Block", thus it needs to divide or unite the files and fulfill the "Block" which is 64MB by default. Therefore, extra metadata will be generated in the Namenode. This space utilization is determined by three space requirement: metadata of directories, files and blocks. In the HDFS which has 64 users:

Table 1 An Example of Metadata Size in Hadoop

	File Vol.	File Size	Dir Meta	Files Meta	Block Meta	Total Meta
DataSet-a	10	100M	10,240	1,250	250	11,790 Bytes
DataSet-b	100	10M	10,240	12,500	250	22,990 Bytes
DataSet-c	1,000	1M	10,240	125,000	250	135,490 Bytes
DataSet-d	5,000	200K	10,240	625,000	250	635,490 Bytes
DataSet-e	10,000	100K	10,240	1,250,000	250	1,260,490 Bytes
DataSet-f	20,000	50K	10,240	2,500,000	250	2,510,490 Bytes
DataSet-g	40,000	25K	10,240	5,000,000	250	5,010,490 Bytes
DataSet-h	80,000	12.5K	10,240	10,000,000	250	10,010,490 Bytes

Directory entry follows the formula as:
144 Bytes + the length of the directory name;
File entry follows the formula as:
112 Bytes + the length of the file name;
Block follows the formula as:
112 Bytes + 24 Bytes * Number of replicas;

In this paper, we assume that the directory name is in an average length of 16 characters, the file name is in an average length of 13 characters, the number of replicas is 2 in the system. The block size is the default value of 64 MB. **Table 1** shows the metadata size for four different file size applications in which data volume is same as 1 GBytes. As it shows, the metadata size will increase together with the number of files or the total file size.

On the other hand, Map-Reduce model requires the Namenode to keep all the metadata in main memory since they are regularly accessed. However, too many of these small files will exceed the addressing capacity within this node and those small file I/Os can flood the networking near Namenode, eventually. Thus, it prevents the performance of the system and leaves the remaining storage capacity unutilized. Moreover, Hadoop has to start new process for every map task. These start-up processes will occupy a great portion of time for the small file I/O application whose execution time of each task is relatively short. In addition, the Datanodes in Hadoop may have large states to checkpoint, they contend for disk and network bandwidth resources, slowing down individual checkpoints.

While for the Fat-Btree database, the metadata needed to be stored for each tuple are limited. During data accessing process, the index pages are required

only for locating the leaf pages stored in each PE . Therefore, it has a high cache hit rate if the index pages are cached in each PE. Because of this high cache hit rate and low maintain cost (Section 2.3), select and update processes in Fat-Btree are much faster than other conventional parallel B-tree structures in 6). In addition, the DBMS is able to achieve high performance by using the index to accelerate join operations.

Since there is a trade-off in Hadoop who inherits the features of high scalability and performance from Map-Reduce while contains the drawbacks mentioned above, we would like to evaluate Hadoop performance under different file I/O applications and compare it with a parallel database implemented with the open source DBMS Postgres and the Fat-Btree index in 20).

4. Experiments

In this section we describe the experiments of Hadoop and our Fat-Btree database. We focus on evaluating data load (initialize) time and data accessing time, as well as system scalability.

4.1 Environmental Setup

For our experiments, we used a cluster of up to 32 nodes. **Table 2** shows our experimental environment.

All the nodes run the Hadoop version 0.20.1 on JDK 1.6.0, and we deploy the system with the default configuration settings, except for changing the replicas number to 1 which is the same as that in Fat-Btree database so as to compare with it later. We do not use multiple replicas because we focus on the file I/O performance in Hadoop here. In addition, we use an individual node as the Namenode in the experiments to ensure that the performance of Datanodes will not be affected.

4.2 Data Load Performance

In this experiment, we load the same datasets as shown in Table 1, from the node's local disk into each system's internal storage to study their performance by examining the time it takes. Although Hadoop has been designed to run on a very large number of nodes, the Datanode number in our experiments is only up to 32 nodes. This is because by our observations, this software is typically used with significantly less instances in current computation clouds. For example, Amazon limits the number of nodes for their Hadoop application to 20 unless the respective customer passes an extended registration request in 26).

Table 2 Experimental Environment

Blade server:	Sun Fire B200x Blade Server
CPU:	AMD Athlon XP-M 1800+ (1.53 GHz)
Memory:	PC2100 DDR SDRAM 1 GB
Network:	1000BASE-T
Gigabit Ethernet Switch:	Catalyst 6505 (720GB/s backbone)
Hard Drives:	TOSHIBA MK3019GAX (30 GB, 5400 rpm, 2.5 inch)
OS:	Linux 2.4.20
Java VM:	Sun J2SE SDK 1.6.0 18 Server VM

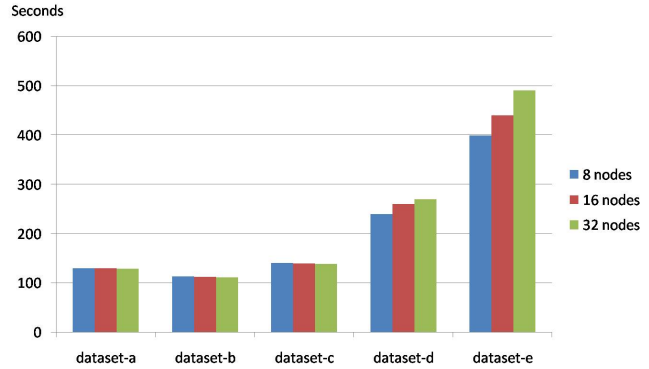

Fig. 3 Data Load Time of Different File Sizes

We first use the dataset-{a, b, c, d, e} in this experiment to load HDFS by the command-line utility provide in Hadoop. The files in these datasets are in plain text manner to enable "wordcount" function available in later experiments.

Figure 3 shows the data load performance of loading the same volume of dataset-1GB. The data loading time is not same from each other if the size per each file is different. The time required for dataset-{a, b, c} are almost same. The little difference may lie in the different file split and combination cost when fill the data into blocks. For the dataset-{d, e} Hadoop consumed much larger time. This reason may be that Hadoop forces each task to run in an individual process, thus each task has to redo the same initialization in its process. When

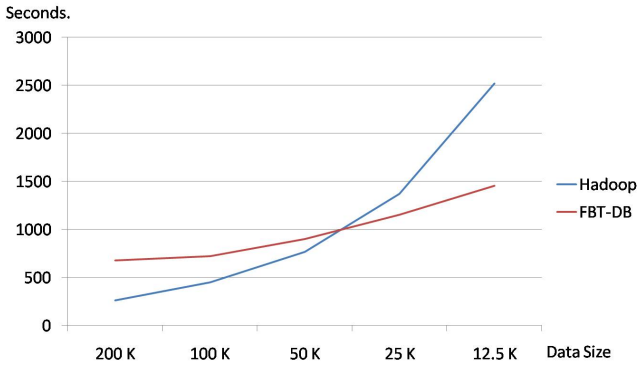

Fig. 4 Data Load Time Comparison

the file number is increasing, the process start-up overhead will soon come to be un-ignorable.

Now, we use dataset-{f, g, h}, a small file dataset to make a comparison between Hadoop and Fat-Btree database in the data load experiment.

In this experiment, Fat-Btree DBMS loads the data by executing the LOAD SQL command sequentially from one node. Then the Fat-Btree evenly distributes these tuples across the cluster automatically. The tuples which are inserted into Fat-Btree DBMS contain the same content as the small files that loaded into Hadoop. **Figure 4** shows the time needed to load the same volume of datasets both in Hadoop and Fat-Btree DBMS. As it shows, the data load time is increased when the individual data size is decreasing in both systems. Note that when the data (file/tuple) size is larger than 100K, the load time in both Hadoop and Fat-Btree DBMS do not grow obviously, because a great portion of cost in writing one piece of data is the file I/O cost. However, when the data size becomes smaller, which means the file I/O cost in each process decreased, the extra cost turns to be dominate. Therefore, the load time in Hadoop soon overcomes that in Fat-Btree DBMS because the Hadoop has a larger process initialization, data combination and metadata management costs in storing the same volume of data, while these extra costs may be unobvious in the parallel databases, which have optimized and sophisticated process and storage management methods.

In a short conclusion, to load the dataset from local disk into Hadoop's virtual disk, lots of data input/split tasks will be launched as several individual processes. If the input file is in a small size (e.g. less than 50K in this experimental environment), the load performance of Hadoop will decrease dramatically, and underperform the Fat-Btree DBMS.

4.3 Data Access Performance

Now, we study the data access performance of Hadoop, including the throughput and scalability, and make a comparison with Fat-Btree DBMS.

Firstly, we modify the demonstration Map-Reduce program "wordcount" in Hadoop source code. To test Hadoop I/O performance: The Map function in "wordcount" is modified to read all the words in the small files on HDFS and write record of each word into an output file as the (key, 1) pairs, there is no calculation function in Map to get the summary of each word's appearance times. And then, we remove the Reduce function in the program, thus the output generated by each Map instance is the final output of the program. Our purpose of these modifications is to get the time required by Hadoop to read and write every word in all the files in HDFS while eliminate the extra cost of calculating, grouping and shuffling. Thus the execution time of this program is mainly the time for the file I/O process in Hadoop, then we can compare this time with that required by the Fat-Btree DBMS.

On the other hand, we use the SQL Command like: SELECT * FROM TABLE WHERE ID == 'X'; to fetch all the contents from the tuples in Fat-Btree DBMS and use the SQL Command like: UPDATE TABLE SET TEXT = 'STRING' WHERE ID == 'X' to overwrite the same data. Each node has 4 client threads to execute these commands in parallel to read and write all the data in the database.

As shown in **Fig. 5**, Hadoop takes several hours to finish the "wordcount" transaction, while Fat-Btree outperforms Hadoop by a factor about 400, which only needs several minutes to read and write all the data. This is because the start-up costs for the small file I/O applications are dominated in the execution time. On average, each task in Hadoop needs about 20 seconds to start and run at full speed.

In addition, Fig. 5 also shows the different scalabilities of Hadoop and Fat-Btree

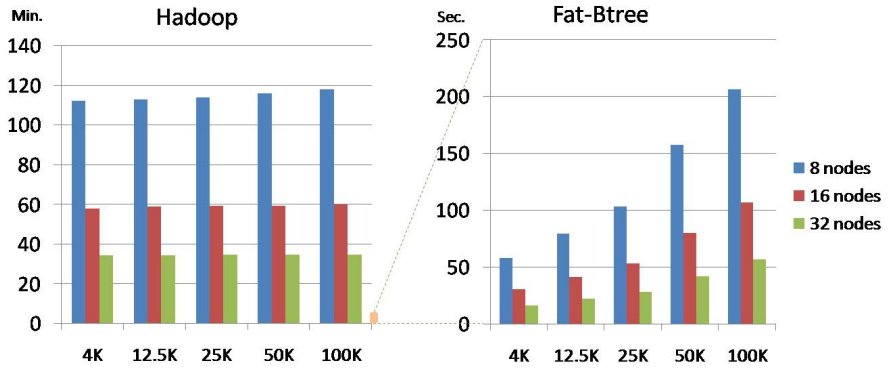

Fig. 5 Data Access Time in Hadoop and Fat-Btree

systems. We find that when the node number is increased by twice from 8 to 32, the scalability of Hadoop declines from 96% to 85% compared to that of Fat-Btree which only decreases slightly. This is because Hadoop use the job tracker to coordinate Datanode activities, and its overhead will increase as more nodes are added into the system. However, Fat-Btree DBMS does not contain these centered nodes, thus the potential bottlenecks are eliminated from the system. The reason for the slight decline may be the raise of communication cost within the Fat-Btree when the node number increasing.

In a short conclusion, for the small file I/O applications, Hadoop's performance decreased dramatically because the overhead in process start-up. It may cause every task to wait 20 seconds on average before they are executed in full speed. On the other hand, Fat-Btree DBMS does not suffer this problem and achieve a better scalability.

5. Related Works

As an open source software which implements a framework of Map-Reduce, Hadoop has been embraced by lots of data-intensive distributed applications both in commercial and academic societies. However, Hadoop seems only to be able to run large-scale analyses on big data. Almost all the applications adopt Hadoop engine for the computation on huge datasets 17)–18), because the performance for the application on small or middle size dataset in Hadoop is not as attractive as DBMS. A lot of researches that focus on these problems have been proposed. For instance, 19) optimizes the Hadoop in its branch project of HPMR, which focuses on cutting off the cost in the "Reduce" process with the High Performance Map-Reduce Engine. 17) proposes a method to reduce the metadata for the small files in Hadoop by compressing the small data which has some kind of semantic relations in the specific applications together. In 1), the small file metadata problem is slightly addressed in the original web implementation of Hadoop. However, there seem no effective solutions in these former researches for improving the small file I/O performance in Hadoop as discussed in this paper.

Therefore, recent researches, which use Hadoop as its execution engine, adopt a light database system into the system for organization the intermediate results and trial datasets, as in 21)–24). These systems try to bring Map-Reduce ideas together with DBMS system and aim to integrate declarative query constructs from the database community into MapReduce-like software to allow better code reusability, data independence and automatic query optimization. Different from above interface level solutions, HadoopDB in 7) integrates Hadoop with Postgres in systems-level, which may be the first practice that builds a real hybrid system of Map-Reduce and Parallel DBMS. What is more, this kind of hybrid Map-Reduce/DBMS system is looked upon as an optimal solution for parallel processing systems in recent future 15).

However, HadoopDB does not in general match the performance of parallel database systems especially in data selection. Since HadoopDB use the databases to store all the data sources similar to data blocks in HDFS, the large size of data used in 7) may be one of the reasons that hinders the data access efficiency of HadoopDB. Therefore, it is very important to choose the better storage layer, database or HDFS layer, dynamically in the future hybrid Map-Reduce/DBMS systems to store the files in the applications. To make this decision, the comparison between Hadoop and parallel DBMS in their file I/O performance on the datasets of different data size is needed for the future system design information.

As far as we know, there seems no former work that gives the comparison. In recently, there is a similar comparison work as ours 8), which compares Hadoop

with other two parallel DBMSs is proposed. However, they focus on the analyses of different storage mechanism (row-based/column based) and aggregation performance in the vertical/horizontal partition DBMS with Hadoop. In addition, the size of individual data used in their experimental datasets is much larger than that of ours. What is more, the tranditional hash and clustered index used in these parallel DBMSs may hinder the database systems scalability. Note that the data load performance experiment result is hence different with ours; the scalability comparison result between our parallel DBMS and Hadoop is also different from that declared in 8).

6. Conclusion

In this paper, we focus on comparing the Hadoop, which is an implementation of map-reduce paradigm, with a parallel database system that was developed earlier at Tokyo Institute of Technology. We have taken eight different datasets to examine the data loading, accessing and modification cost of these two system especially for small file I/O. These comparison results could be a note for the information of future hybrid Map-Reduce and Parallel DBMS design.

Our results show that, Hadoop has significant overheads due to task initialization; the Namenode appears as the bottleneck when the system scales up, especially for handling the small files. Our parallel DBMS shows its higher performance, over a factor of 400 in the small file I/O performance, and shows its better scalability than that of Hadoop. Note that some of the system features in both Hadoop and DBMS are no longer the same as they used to be in former researches which mainly considered the data in large file size. As there is a great contrast in file I/O performance of Hadoop and parallel DBMS when handling the small size of data, a flexible storage layers mechanism should be considered in the design of the future hybrid Map-Reduce/DBMS systems.

Acknowledgments Part of this research was sponsored by CREST of Japan Science and Technology Agency (JST), and MEXT via a Grant-in-Aid for Scientific Research on Priority Areas #19024028 and #22240005.

References

1) http://www.cloudera.com/blog/2009/02/02/the-small-files-problem.
2) J. Dean and S. Ghemawat. :"MapReduce: Simplified Data Processing on Large Clusters", *In USENIX Symposium on Operating Systems Design and Implementation*, OSDI' 2004.
3) Wu, S. and Kemme, B. 2005. Postgres-R(SI): Combining Replica Control with Concurrency Control Based on Snapshot Isolation. *In Proceedings of the 21st Int'l Conf. on Data Engineering*, ICDE '2005, April 05 - 08, Washington, DC. pp. 422-433
4) H. Boral, W. Alexander, L. Clay, G. Copeland, S. Danforth, M. Franklin, B. Hart, M. Smith, and P. Valduriez, :Prototyping Bubba, a highly parallel database system, *IEEE TKDE*, vol. 2, no. 1, pp. 4-24, 1990
5) D. J. Dewitt, S. Ghandeharizadeh, D. A. Schneider, A. Bricker, H. I. Hsiao, and R. Rasmussen, :"The Gamma database machine project", *IEEE TKDE*, vol. 2, no. 1, pp. 44-62, 1990.
6) H. Yokota, Y. Kanemasa, and J. Miyazaki, :"Fat-Btree: An update conscious parallel directory structure", *ICDE '99*, IEEE Computer Society, Mar. 1999, pp. 448-457
7) A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Rasin, and A. Silberschatz, :"Hadoopdb: An architectural hybrid of mapreduce and dbms technologies for analytical workloads", *VLDB'09 Proceedings of the 2009 VLDB Endowment*.
8) A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. Dewitt, S. Madden, and M. Stonebraker, :"A comparison of approaches to large-scale data analysis" in *Proceedings of the 2009 ACM SIGMOD International Conference*, ACM, June 2009
9) "Hadoop," http://hadoop.apache.org/.
10) C'esar Augusto S'anchez Baquero, :"Performance test of Hadoop and iRODS Distributed Storage Systems", *Seminario De Invesigation III*, Mayo 18 De 2009
11) J. Delmerico, N. Byrnes, A. Bruno, M. Jones, S. Gallo, and V. Chaudhary, :"Comparing the Performance of Clusters, Hadoop and Active Disks on Microarray Correlation Computations", *the 16th IEEE International Conference on High Performance Computing*, HiPC '2009, Cochin, India.
12) http://dev.mysql.com/doc/refman/5.1/en/overview.html
13) E. Pacitti, M. T. Ozsu, and C. Coulon, :"Preventive multi-master replication in a cluster of autonomous databases", *In Euro-Par*, 2003, pp. 318-327.
14) http://slony.info/documentation/failover.html
15) M. Stonebraker, D. Abadi, D. J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin, :"Mapreduce and parallel dbmss: friends or foes?" *Commun. ACM*, vol. 53, no. 1, pp. 64-71, 2010
16) Philip Carns, Sam Lang, Robert Ross, Murali Vilayannur, Julian Kunkel, and Thomas Ludwig. :"Small-File Access in Parallel File Systems", *Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium*, April 2009
17) Xuhui Liu, Jizhong Han, etc. :"Implementing WebGIS on Hadoop: A Case Study of Improving Small File IO Performance on HDFS", *IEEE Cluster*, New Orleans, LA, September 1, 2009

Vol.2010-DBS-150 No.7
Vol.2010-IFAT-99 No.7
2010/8/4

18) Leo, Simone; Santoni, Federico; Zanetti, Gianluigi, :"Biodoop: Bioinformatics on Hadoop", *International Conference on Parallel Processing Workshops, 2009*, ICPPW '09, vol., no., pp.415-422, 22-25 Sept. 2009
19) Sangwon Seo; Ingook Jang; :"HPMR: Prefetching and pre-shuffling in shared MapReduce computation environment", *Cluster Computing and Workshops*, 2009.
20) Yuta Namiki, Kota Kanbe, Dai Kobayashi, and Haruo Yokota, :"An approach of using a parallel B-tree structure, Fat-Btree, in PostgreSQL for distributed retrieval", *DBSJ Letters*, Vol6, No.2, pp.61-64, Sept. 2007
21) R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. :"Scope: Easy and efficient parallel processing of massive data sets", *Proc. of International Conference on Very Large Databases*, 2008.
22) C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. :"Pig latin: a not-so-foreign language for data processing", *Proc. of SIGMOD*, 2008.
23) Facebook. Hive. Web page issues.apache.org/jira/browse/HADOOP-3601
24) Thusoo, A. et al. :"Hive: A warehousing solution over a Map-Reduce framework", *Proceedings of the Conference on Very Large Databases*, 2009, 1626–1629.
25) Wang, Feng and Qiu, Jie and Yang, Jie and Dong, Bo and Li, Xinhui and Li, Ying. :"Hadoop high availability through metadata replication" *CloudDB '09: Proceeding of the first international workshop on Cloud data management*.
26) Amazon Web Service LLC. Amazon Elastic MapReduce. http://aws.amazon.com/elasticmapreduce/,2009.