論文 / 著書情報
Article / Book Information

論題(和文)	Evaluation of Multiple Fat-Btrees on a Parallel Database
Title(English)	Evaluation of Multiple Fat-Btrees on a Parallel Database
著者(和文)	Min LUO, Takeshi Mishima, Haruo YOKOTA
Authors(English)	Min LUO, Takeshi Mishima, Haruo YOKOTA
出典(和文)	, , ,
Citation(English)	, , ,
発行日 / Pub. date	2012, 3

Evaluation of Multiple Fat-Btrees on a Parallel Database

Min LUO † Takeshi MISHIMA ‡ Haruo YOKOTA †

† Dept. of Computer Science, Tokyo Institute of Technology 2-12-1 Ookayama, Meguro-ku, Tokyo, 152-8552 Japan

‡ NTT Information Sharing Platform Laboratories 3-9-11 Midori-cho, Musashino-shi, Tokyo, 180-8585 Japan

E-mail: † luomin@de. cs.titech.ac.jp, ‡ mishima.takeshi@lab.ntt.co.jp, † yokota@cs.titech.ac.jp

Abstract Fat-Btree, as well as the following contribution work for its efficient concurrent control, load balancing and data reliability, has been proposed for the high-speed access in parallel database systems on shared-nothing environment. Because these contributions are focus on the index accessing and maintenance efficiency only, a single Fat-Btree structure in a parallel database system is sufficient for all the previous evaluations. However, databases with multiple relations require multiple Fat-Btrees for the parallel accessing. In this work, we introduce the construction of a multiple Fat-Btree index system for multiple relations, a study of the throughput in this system using PostgreSQL based DBMS cluster is provided.

Keyword Fat-Btree, Multiple Relation, Parallel Database

1. Introduction

"Cluster computing" has attracted considerable attentions in high performance and scalable distributed systems research. In these systems, a large number of low-end servers are lined up and work in parallel to act as a smaller set of high-end servers. Many data accessing methods and distributed execution frameworks have been proposed to coordinate the clusters [2, 3, 4, 5, 6, 7].

In these frameworks, Map-Reduce [2] is one of the most famous one and 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. On the other hand, [5], [6], [7] provide many data accessing methods in parallel databases for data processing on the "clusters". Besides the contributions made over the past two decades, many ongoing academic projects are also engaged to provide better performance, scalability and failure tolerance parallel database systems [11], [12], [13], [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, lots of comparisons between these two systems have been carried out for the users' information. For instance, [8] showed that DBMSs on shared-nothing clusters outperform Map-Reduce by a large factor in a variety of tasks. Additional comparisons in [9] showed that the Fat-Btree [7] based parallel DBMSs possess higher scalability and less data loading time than Map-Reduce system as well, which are different observations from [9], especially for small file I/O.

Besides the scalable and efficient data accessing performance, the variant of Fat-Btree, a compound Fat-Btree has been introduced for dynamical access-skew balancing ability [1, 10]. It balances the skew without high-cost data migration or index reconstruction processes as in ordinary parallel DBMSs, but only modifies the data accessing paths to the replicas on other nodes by switching the flag identifier in the compound index.

The previous work [1, 7, 9 10] have verified the high scalability and availability in the Fat-Btree, but these contributions are focus on the index accessing and maintenance efficiency, and only one single Fat-Btree structure is constructed in a parallel database system.

However, databases with multiple relations require multiple Fat-Btrees for the efficient parallel accessing. Furthermore, parallel join-operations could be optimized by the multiple Fat-Btree parallel indices. Therefore, it is important to provide a parallel database of multiple Fat-Btree indices, and study the system performance.

In this paper, we introduce the construction of a multiple Fat-Btree indices system for multiple relations. We also compare and discuss the experimental results of this system with that of the single Fat-Btree system.

2. Background

We briefly review the existing technologies for high scalable and available parallel index and introduce a shared-nothing parallel database based on Fat-Btree index.

2.1 Parallel Indexing Structures

There are two main methods of distributed data accessing. DHT-based methods uniformly map nodes and data objects into a single ID space, and each node is responsible for a specific range of the ID space. On the other hand, B-tree based parallel index is efficient in range-query, but skewed range access may lead to obvious performance degradation, unless the migration of the skewed data and index structure.

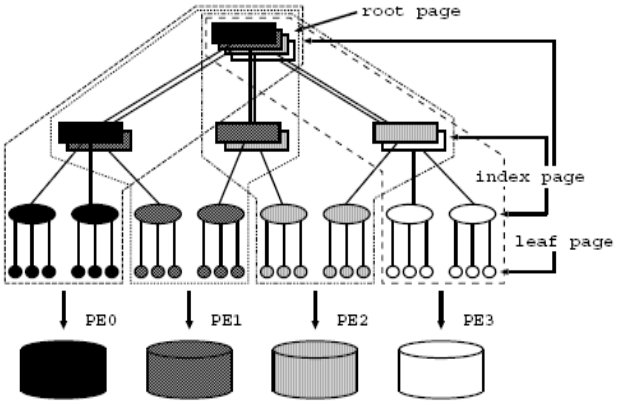

Fig.1 Fat-Btree

The Fat-Btree [7] is introduced to reduce the update maintenance and index migration cost, as a parallel B-tree index. An example of a four-PE Fat-Btree is given in Fig. 1, where multiple copies of index nodes close to the root node with relatively low update frequency are replicated on several PEs, while leaf nodes with relatively high update frequency are distributed across the PEs. Thus the index nodes that require synchronization in different PEs are greatly reduced and it also provides load index migration methods [7] than other parallel Btree structures.

2.2 One Fat-Btree Based System

We introduce the architecture of a proposed one Fat-Btree based parallel database system here. As shown in Fig. 2, each PE in this system contains two main modules. The open-source PostgreSQL DBMS is chosen for the DBMS module and the Fat-Btree parallel index is implemented inside the FBT module.

For the relation with Fat-Btree, its tuples are sorted and every 60 tuples are put into one data-page which is stored in the Page Server. To index these data-pages by Fat-Btree, the min_value of the tuple in a data-page is used as the key_id. As described in Sec. 2.1, the root index node is replicated on all the PEs, and there are redundant intermediate index nodes between any neighbor PEs. Because of these redundant intermediate index nodes, a parent node has pointers to their child nodes in the neighbor PE. Thus, the root node has a point path to any leaf node in any PE. Therefore, a client is able to retrieve any tuples from any PE in the system.

Fig. 2 also shows a query processing flow in this system. The sequence of the red arrow lines illustrates how to handle a remote query. We assume all the queries are querying the Fat-Btree indexed attribute. Their processing sequences are shown as the red ordered arrows in Fig. 2.

a). Clients send query through a socket connection to the Fat-Btree system.

b). A thread pool receives these queries and issues a “SQL Server” for processing each query.

c). “SQL Server” extracts the ‘key’ in the query and send it to “FBT Mgr.” (Fat-Btree manager) through the “Comm. Mgr.” (communication manager).

d). Based on the information at local Fat-Btree, “FBT Mgr.” forwards the query to a remote PE, where the target data is located, through “Comm. Mgr.”.

e). The target PE receives the query, and its local “FBT Mgr.” verifies the ‘key’ is contained by current PE.

f). Start query processing in the DBMS module.

f1). If the target page is already in the PGSQL buffer, return the tuple in the page.

f2). If the target page is not in the PGSQL buffer, load the target page from Page Server by using the ‘key’ of target tuple.

f2.1). traverse Fat-Btree to find the leaf node that contains the ‘key’

f2.2) get the data page No. in the index leaf

f2.3) fetch the page in Page Server by the No.

f2.4) load the page into PGSQL buffer.

f2.5) return the target tuple in the page

f3). Increase the accessing account of the page

If the above query is for a non-Fat-Btree indexed relation, DBMS module follows the query handling process in original PostgreSQL.

Note that, in the step-d) above, remote queries have to be transmitted between PEs by the “Comm. Mgr” through socket connections. This transmission overhead grows when the number of PEs or clients increases.

3. Multiple Fat-Btrees System Structure

3.1 System Structure Discussion

As described in Sec. 2.2, in the Fat-Btree index module, the independent thread pool handles query issuing, the Comm. Mgr. handles the remote query transmission, the

FBT Mgr. handles Fat-Btree traversing, and the Page Server stores the Fat-Btree indexed data pages. These resources are competitive even within the one Fat-Btree system, when there are multiple threads processing client queries concurrently. Therefore, we duplicated them when constructing multiple Fat-Btrees.

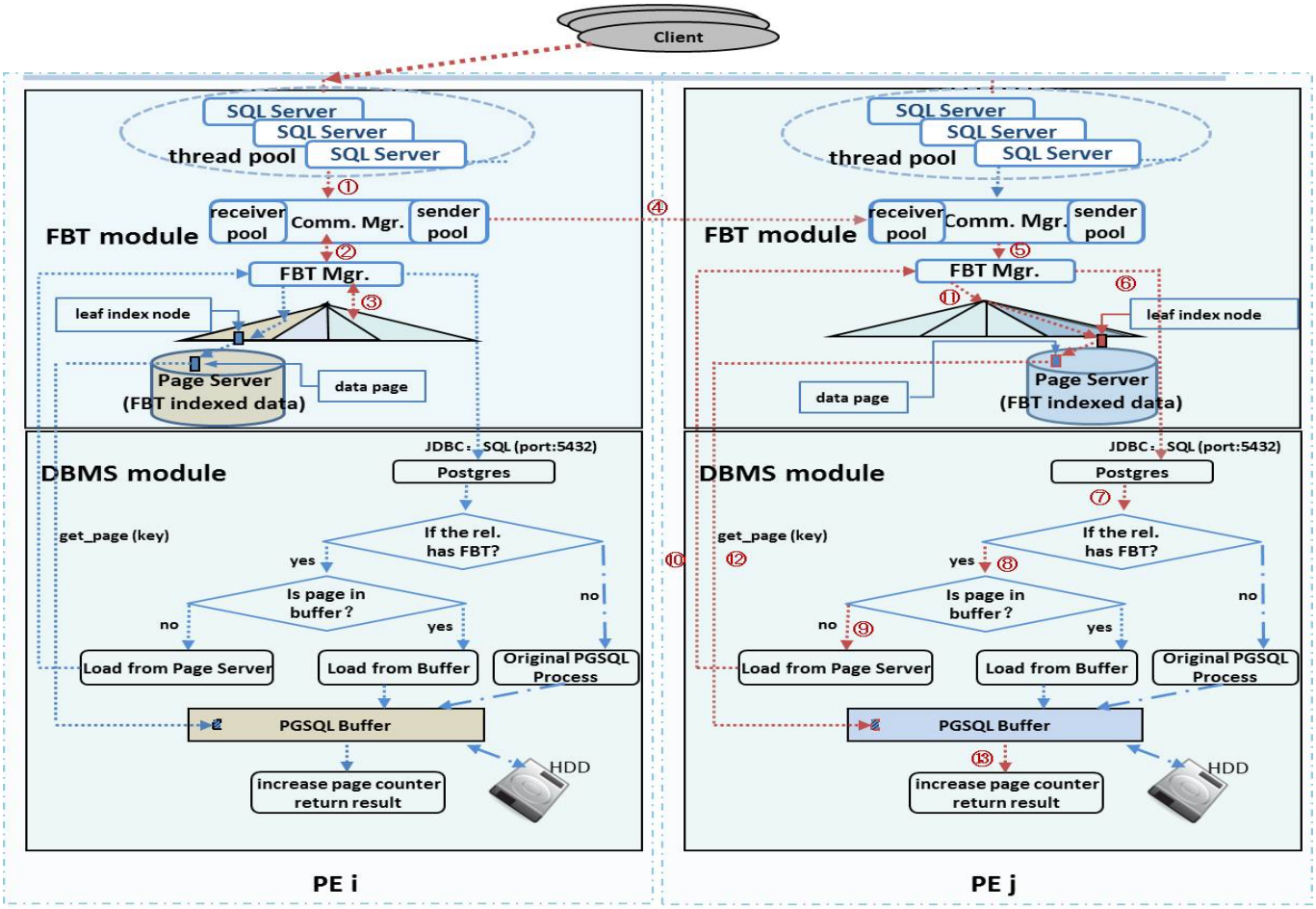

Fig.2 One Fat-Btree System Structure

Because it requires data I/O between PGSQL buffer and hard disk when retrieve a non-Fat-Btree indexed relation. While indexing the relation by a Fat-Btree in the Multi-FBT system will load the relation into the Page Server in memory. Therefore, the Multi-FBT system improves the data retrieving efficiency by avoiding the disk I/Os. In addition, Fat-Btree index will also provide efficient access to its data pages stored in the Page Server.

3.2 Multiple Fat-Btrees System Structure

In this multiple Fat-Btree system structure, relations are range partitioned across the PEs, and they have their own FBT modules on every PE to manage their data tuples.

Fig. 3 gives an image of two PEs in our system structure. The Page Servers ‘m’ on each PE stores the data of relation ‘m’ that is partitioned on its PE. The Fat-Btree for relation ‘m’ is constructed to index the data pages stored on all these Page Servers ‘m’.

In the DBMS module, the PostgreSQL source code is modified. The index_ids and relation_ids of the Fat-Btrees and Fat-Btree indexed relations are recorded in PostgreSQL. A Postgre instance searches the local buffer based on the relation_id first. If the target data page is in the PGSQL buffer, the Postgre returns the target tuple in this buffered page. Otherwise, Postgre starts to load the page from the corresponding Page Server. The correct FBT Module is chosen by mapping the relation_id to the corresponding index_id. And the data page is retrieved from the target Fat-Btree’s Page Server, through the correct socket connection to the FBT Module..

4. Experimental Comparison

In this section, we provide the evaluations of the proposed multiple Fat-Btree parallel database. We check its throughput scalability when the number of clients in the cluster scales. Then we examine the efficiency of a

single Fat-Btree in the Multi-FBT system by comparing it with that of the single Fat-Btree in the original Single-FBT system. At last, we check the Multi-FBT system scalability when the number of PEs scales.

4.1 Environmental Setup

We initialize two FBT relations in the multiple FBTs system and one FBT relation in the original single FBT system. Each relation has a 10,000 tuples on each PE, and each tuple has 134 bytes. Each PE receives simultaneous requests from its client nodes; the key in each request is generated randomly within the data range. For instance, in a four-PEs configuration, these queries are: key = random (1, 40000); select* from table X where id = key; update table X set value += 1 where id = key. In addition, in one testing course, each client sent 20 queries in series. The tps (throughput per second) value in our experimental result is the average tps result of 20 testing courses.

In addition, on every PE, the number of SQL Server threads is set to the same number of clients per PE, thus all the client threads are processed in parallel, and the requests in one client thread are processed in serial.

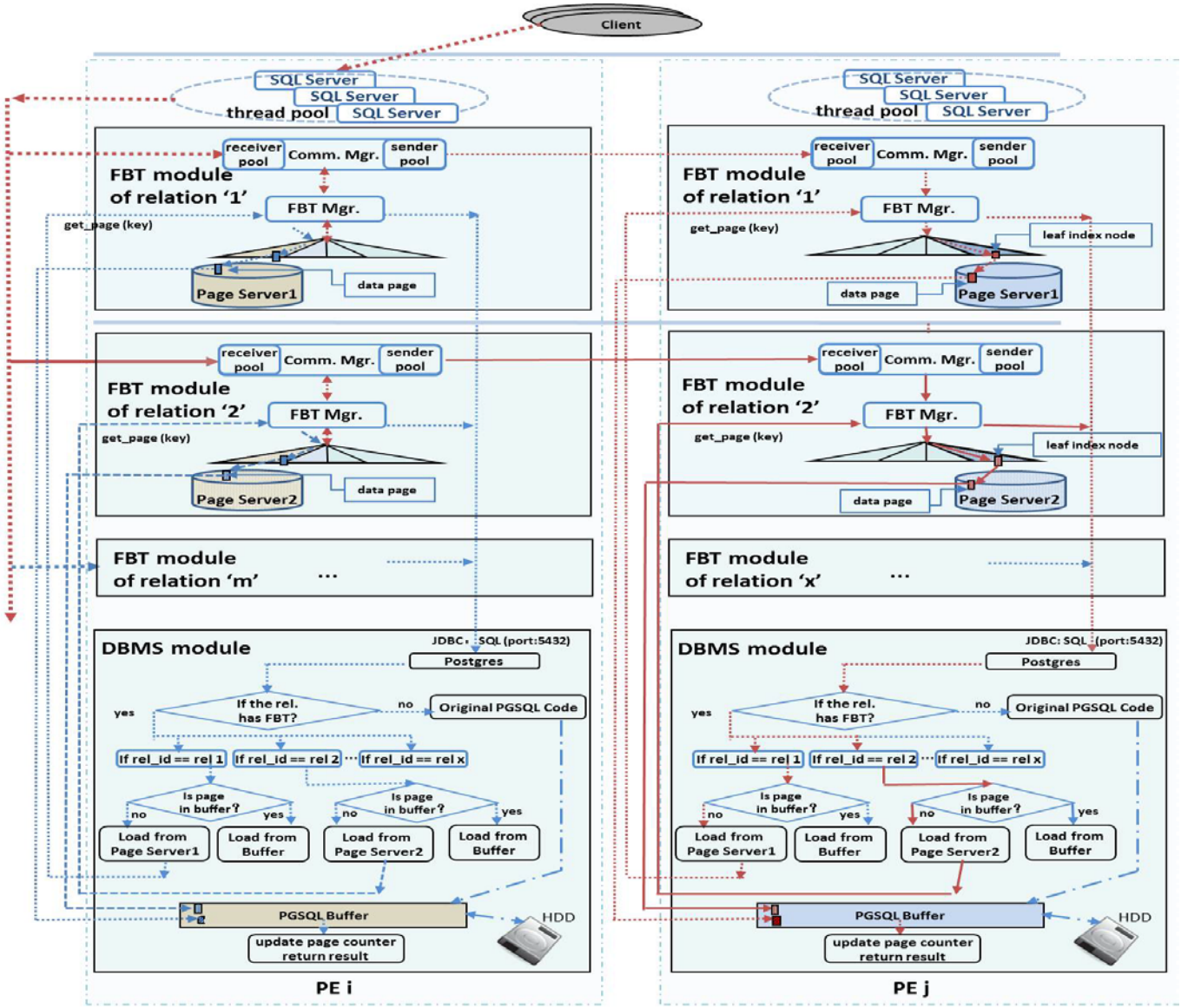

Fig.3 Multiple Fat-Btrees System Structure

TABLE 1
EXPERIMENTAL ENVIRONMENT 1

Process Type:	Intel Xeon E5620
Cores:	4*2
Core Frequency	2..4GHz
Hard Drives:	1TB, (Model:MB1000EBNCF)
Memory:	24GB (4GB DDR3 * 6)
OS:	Ubuntu11.10
Java VM:	Sun J2SE SDK 1.6.018 Server VM
Network:	1000BASE-T

4.1 Evaluation Results

We provide the experimental results to examine the Multi-FBT system efficiency and scalability by evaluating its throughputs. There are up to 8 PEs in our cluster, each PE is consisted of the same environment in Table 1.

Fig. 4 shows the experiment results of all-select & all-update throughput in the Multi-FBT system and original single-FBT system. We run the experiments on 4-PEs and 8-PEs scales in our cluster. The 'x'-axis shows the number of clients on each PE. For example, value '16' means there are 16 clients on one PE querying all the relations evenly in the system. Thus, for the Multi-FBT system of this experiment (two Fat-Btree relations), each relation has 8 clients on each PE.

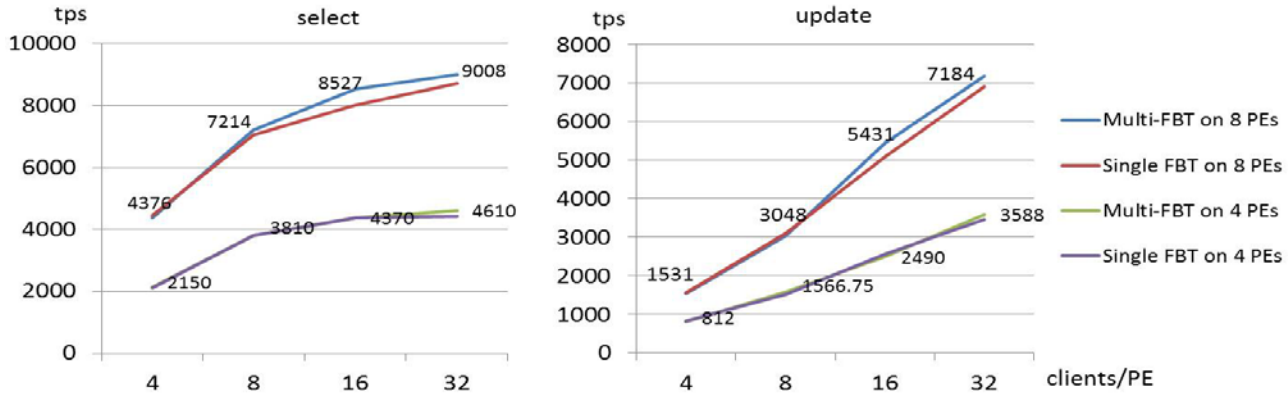

Fig. 4 Comparison of Multiple-FBT vs. Single-FBT system

In the 4-PEs scale test, the throughput of both systems is almost the same, which verifies the extension of another Fat-Btree index does not introduce obvious overhead into the original single-FBT system. We note the throughput of all-select saturated and reached plateau while the scalability of all-update also declined in both systems, when increasing the number of clients. This is because the more number of clients is being served in parallel; the more query transfer messages have to be transmitted and processed between PEs in one parallel index.

In the 8-PEs scale test, the Multi-FBT has higher throughput. Because when the system scales, the number of remote queries also increases. Compared with the single-FBT system, there are two Fat-Btree modules in the Multi-FBT system to share these remote queries transmission. In addition, because the number of clients per each relation in Multi-FBT system is times less than that in Single-FBT system, there are much less conflict update in index nodes modification. Therefore, the Multi-FBT has better performance when system scales up.

In addition, we compare the extended Fat-Btree indices in the Multi-FBT system to examine their same efficiency. Fig. 5 shows the results on our 8-PEs cluster. In this experiment, all the clients in the Multi-FBT system query only one of the relations in the same testing. The results illustrate that the different FBT modules in Multi-FBT system has no obvious difference in their efficiency.

Because the cluster used in these experiments has only 8-PEs. We used another low-end cluster in our lab to examine the system scalability when system scales up to 16 PEs. Its environment is shown in Table 2.

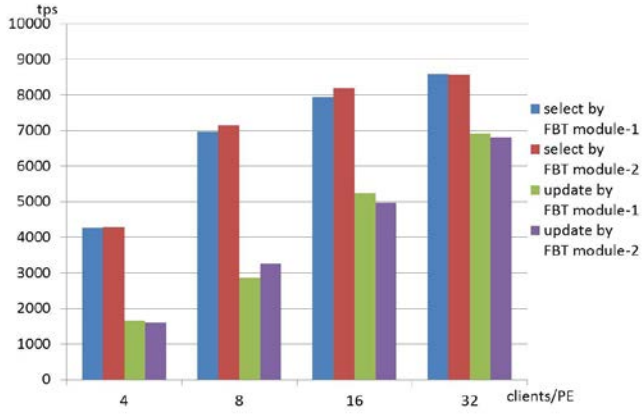

Fig. 5 Comparison of the extended indices

TABLE 2
EXPERIMENTAL ENVIRONMENT 2

Blade server:	Sun Fire B200x Blade Server
CPU:	AMD Athlon XP-M 1800+ (1.53 GHz)
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 1.6.018 Server VM

In this experiment, we simulate 16 clients on each PE, and each of the two Fat-Btree indexed relations will be queried simultaneously by 8 of them. Experimental result graph in Figure 6 illustrates that the multiple Fat-Btree system has over 90% scalability when the number of PEs is doubled from 4 to 16 in both all-read & all-update transactions.

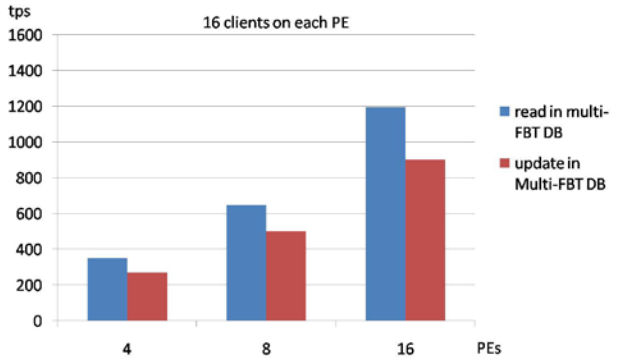

Fig. 6 Scalability Evaluation

In conclusion, we have verified the better throughput and high scalability of the proposed Multi-FBT system by scaling both the number of PEs and clients in the experiments. The multiple Fat-Btrees provides higher performance and scalability than that of the previous single Fat-Btree index system.

5. Future Work

In this paper, we have introduced the architecture of a multiple Fat-Btree based parallel database system. Our evaluation results have shown the proposed Multi-FBT system has higher salability than the single-FBT system when the number of PEs or clients scales up.

With the multiple Fat-Btree parallel indices, parallel join-operation is able to be optimized in the Multi-FBT system. [16, 17] have shown the parallel B-tree index is able to provide the best parallel join performance than other well-known non-parallel-Btree based join algorithms. Because the Fat-Btree has better efficiency than the traditional parallel Btree used in [17], we believe this Multi-FBT system is able to provide high join efficiency. We tend to provide the parallel-join comparisons with other parallel databases in our future work.

Reference

[1] M. Luo, A. Watanabe, and H. Yokota, “Compound treatment of chained declustered replicas using a parallel btree for high scalability and availability”, in Proc. of the 21st international conference on Database and expert systems applications (DEXA 2010), Vol.6262, pp. 49-63 of LNCS. Springer, 2010

[2] Dean and S. Ghemawat. “MapReduce: Simplified Data Processing on Large Clusters”, In *OSDI ’04*,.

[3] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. “Bigtable: A distributed storage system for structured data”, In OSDI, 2006.

[4] S. Ghemawat, H. Gobioff, , and S.-T. Leung. “The Google file system”, In SOSP, 2003.

[5] 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.

[6] 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.

[7] H. Yokota, Y. Kanemasa, and J. Miyazaki, “Fat-Btree: An update conscious parallel directory structure”, in ICDE '99. IEEE Computer Society, Mar. 1999, pp. 448.457.

[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 SIGMOD, 2009.

[9] M. Luo and H. Yokota, “Comparing hadoop and fat-btree based access method for small file i/o applications,” in WAIM ’10: Proc. of the 11th Web-Age Information Management Int’l Conference, July, ser. LNCS, vol. 6184. Springer, 2010.

[10] M. Luo, and H. Yokota, “An Evaluation on Dynamic Access-Skew Balancing Performance of Compound Parallel Btree for Chained Declustering Parallel Systems”, in Proc. of the 3rd Forum on Data Engineering and Information Management (DEIM), Feb. 2011

[11] http://dev.mysql.com/doc/refman/5.1/en/overview.html

[12] Wu, S. and Kemme, B. 2005. “Postgres-R(SI): Combining Replica Control with Concurrency Control Based on Snapshot Isolation”, in Proc. of the 21st Int’l Conf. on Data Engineering, ICDE ’2005, April 05 – 08 Washington, DC. pp. 422-433

[13] E. Pacitti, M. T. Ozsu, and C. Coulon, “Preventive multi-master replication in a cluster of autonomous databases”, in In Euro-Par, 2003, pp. 318-327.

[14] http://slony.info/documentation/failover.html

[15] A. Gupta, D. Agrawal, and A. El Abbadi, “Approximate range selection queries in peer-to-peer”, in Proc. Conf. Innovative Data Systems Research (CIDR), 2002.

[16] Jianzhong Li, Hong Gao, Jizhou Luo, Shengfei Shi, and Wei Zhang. “InfiniteDB: a pc-cluster based parallel massive database management system”, in Proc. of the 2007 ACM SIGMOD Int’l Conf. on Management of data (SIGMOD '07). ACM, New York, NY, USA, 899-909.

[17] Jianzhong Li, Wenjun Sun, and Yingshu Li. “Parallel Join Algorithm based on Parallel B+-trees”, in Proc. of the 3rd Int’l Symposium on Cooperative Database Systems for Advanced Applications (CODAS '01). IEEE Comp. Society, Washington DC, USA, 178-.,.