A Structure of Update-conscious Directories for Parallel Database Systems

Yasuhiko Kanemasa

School of Information Science,
Japan Advanced Institute of Science and Technology

February 13, 1998

Keywords: Parallel Database, Directory Updating, Parallel Index, Parallel B-tree, Fat-Btree.

1 Introduction

In shared-nothing parallel database systems, retrieval and update operations are executed in parallel on each PE where data they access is stored. Therefore, load balancing among PEs results in improvement of processing performance, and data partitioning strategies for load balancing become very important[1, 3]. There are many strategies of partitioning data, such as hash partitioning or range partitioning[2]. However, the hash partitioning cannot cope with range queries which the range partitioning can, and with clustering data accesses. On the other hand, the range partitioning needs very large cost to balance loads when a data skew occurs as a result of updating data. In order to solve these problems, there have been a study using a parallel B-tree for data partitioning[4]. By using a parallel B-tree, we cannot only solve the weak points of both hash and range partitioning strategies, but also speed up data access. But in the previous study on parallel B-trees, there are problems of updating the parallel directory. If all PEs have copies of the whole directory for access distribution, simultaneous update accesses to all PEs reduce throughput of the system. On the other hand, if the directory is placed in one PE, centralized accesses to the PE cause a process bottleneck.

In this study, a new method of structuring a parallel directory, *Fat-Btree*[6], is adopted. This study examines characteristics of the Fat-Btree, compares the method adopting the Fat-Btree with the ordinary method copying the whole B-tree to all PEs by probability based performance analyses, and shows that the Fat-Btree structure is better in respects of both throughput and response time. This study also examines a way for implementing Fat-Btree, and considers about the alternative ways to implement which are caused by the difference between Fat-Btree and ordinary B-trees.

2 Fat-Btree

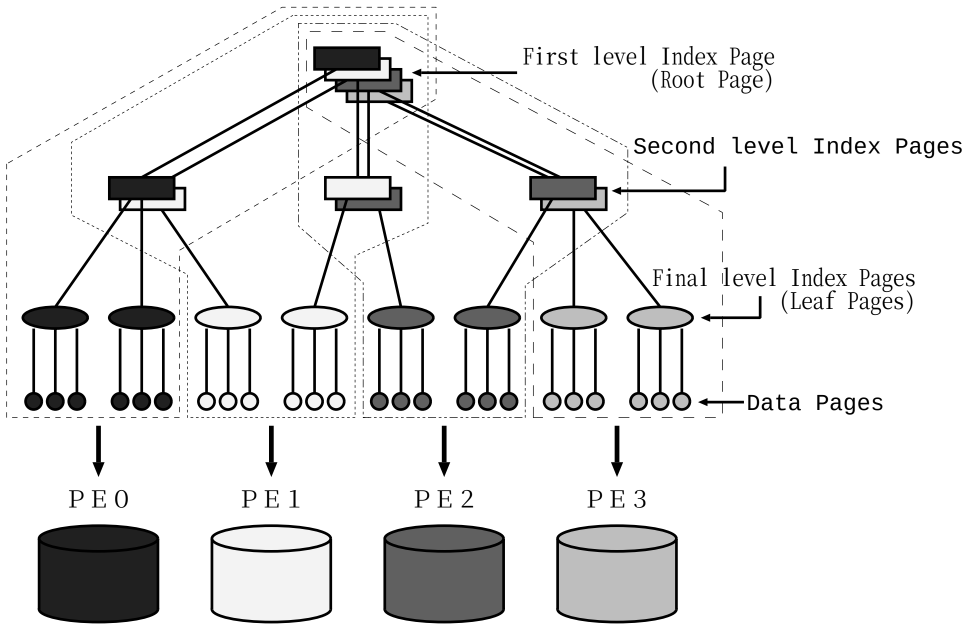

Figure 1: Fat-Btree structure

In Fat-Btree structure, leaf pages (final level index pages) of B-tree are distributed uniformly to all PEs. With regard to index pages (except leaf pages) of B-tree, the pages which are recursively above the leaf pages stored in the PE are stored in the same PE. Hence, in the Fat-Btree structure, the subtree from the root page to the leaf pages is stored to each PE (Figure 1).

In Fat-Btree structure, higher level index pages included in many paths to leaf pages are copied and distributed to PEs having descendent pages. Thus the root page is copied to all PEs. However, index pages modified in a directory update are mostly lower level pages, and those pages are copied to a less PEs. So, the processing cost of directory update in the Fat-Btree method is expected to be lower than one for the method copying the whole B-tree to all PEs. Moreover, since whole path from the root page to a leaf page is stored in one PE, data retrieval can be done in parallel on each PE. It can avoid concentrating processes on a few PEs.

3 Probability based Performance Analyses

In order to show usefulness of Fat-Btree, this study compares the method adopting Fat-Btree with the ordinary method copying the whole B-tree to all PEs[4] (say the whole B-tree copying method) using probability based performance analyses. In the analyses, the numbers of page accesses and communication messages both which are necessary for one READ or WRITE operation, are calculated from expected value of the number of copies per an index page, rate of incidence of splitting a page, and so on. From these numbers, the total processing time and response time per one operation are calculated. Throughput and response time of the system are calculated from them.

The analysis results indicate that the Fat-Btree is mostly better than the whole B-tree copying method in respects of throughput, unless memory caches are not used, the number of PEs is not large, and the directory updates are few. In particular, the larger the numbers of PEs and tuples grow, and the higher the ratio of write operations increases, on a larger scale the Fat-Btree improve the whole B-tree copying method in respects of throughput. If the number of PEs grows higher, the whole B-tree copying method needs high cost for simultaneous update accesses to all PEs, so it reduces system throughput. On the other hand, the Fat-Btree restricts the PEs involved in the directory updating to a few PEs, so it has few simultaneous update accesses to many PEs, and it improve the system's throughput linearly as the number of PEs grows. Moreover, the Fat-Btree stores fewer directory pages on a PE, therefore it increases hit ratio of cache memory, and it is better than the whole B-tree copying method also in respects of response time.

4 Implement of Fat-Btree

In Fat-Btree structure, since some child pages of a index page are in other PEs and have many copies in many PEs, it is a problem that how to link the parent index page with the child index page. One of the good solutions for the problem is that, physical page number of a child index page stored in the same PE with parent index page is recorded into the parent index page, ones of child pages stored in other PEs are recorded into another pages, and the additional pages are linked from the parent index page.

At making initial Fat-Btree structure from a relation, it is a problem that how to make the respective different directories in each PE. One of the good solutions for the problem is that, every PE makes the identical Fat-Btree structure simultaneously from the same relation, and then each PE deletes respective unnecessary pages in it.

In Fat-Btree structure, an access path spreads over a few PEs, so the page locking protocol is an important issue. In the case of READ, the Fat-Btree structure had better adopt a modified B-link structure[5]. Then a lock on the parent page can be released before a lock on the child page is acquired when the query is taken over between PEs. In the case of WRITE, an optimistic page locking protocol using IX locks suits to Fat-Btree. In this protocol, an updater traces access path to leaf page using IX locks, and if the updater updates the directory as a result of a write process in the leaf page, the updater redos its tracing from the root page using SIX locks.

5 Conclusions

The results of our probability based performance analyses indicate that the larger the numbers of PEs and tuples grow, and the higher the ratio of write operations increases, on a larger scale the Fat-Btree improve the ordinary method copying the whole B-tree to all PEs in respects of throughput. Moreover, our Fat-Btree method improves response time as well as the throughput. We presented about these analysis results in IEICE Special Interest Group of Data Engineering[7].

This study also examined the way to implement Fat-Btree, and considered about structures of an index page, methods for making the initial Fat-Btree structure, and page locking protocols. We will present about these consideration about implement Fat-Btree in Data Engineering Workshop(DEWS'98).

Future works could involve: we need to implement Fat-Btree indeed, and to do more detailed performance analyses that take account of waiting time for locking and real number of copy pages.

References

[1] G. Copeland and W. Alexander and E. Boughter and T Keller, "Data Placement In Bubba", Proc. of ACM SIGMOD Conf., pp.99–108 (1988).

[2] D. DeWitt and J. Gray, "Parallel Database Systems: The Future of High Performance Database Systems", CACM, Vol. 35, No. 6, pp.85–98 (1992).

[3] S. Ghandeharizadeh and D. J. DeWitt, "Hybrid-Range Partitioning Strategy: A New Declustering Strategy for Multiprocessor Database Machines", Proc. of 16th VLDB Conf., pp.481–492 (1990).

[4] B. Seeger and P. Larson, "Multi-Disk B-trees", Proc. of ACM SIGMOD Conf., pp.436–445 (1991).

[5] V. Srinivasan and M. J. Carey, "Performance of B-Tree Concurrency Control Algorithms", Proc. of ACM SIGMOD Conf., pp.416–425 (1991).

[6] H. Yokota, J. Miyazaki, Y. Tsuchiya, "Parallel Processings in Parallel Active Database Engine (Parade)", in Japanese, Proc. of Matsue Workshop on Advanced Database, pp426–433 (1996).

[7] Y. Kanemasa, J. Miyazaki, H. Yokota, "A Structure of Update-conscious Directories for Parallel Database Systems", in Japanese, Technical Report of IEICE DE97–77(AI97–44) (1997).