ABSTRACT

Phase change memory (PCM) as a newly developed storage medium has many attractive properties such as non-volatility, byte addressability, high density and low energy consumption. Thus, PCM can be used to build non-volatile main memory databases. However, PCM’s long write latency and high write energy bring challenges to PCM-based memory systems. In this paper, we propose an improvement over the B+-tree for PCM. Particularly, we consider the read/write tendency of leaf nodes. For write-intensive leaf nodes, we use an overflow-node technique to reduce PCM writes, while for read-intensive ones, we adjust the tree structure to remove overflow nodes to improve read performance. Our experimental results suggest that our proposal outperforms the traditional B+-tree and the overflow B+-tree.

CCS Concepts

• Information systems → Information storage systems
• Information storage technologies → Storage class memory.

Keywords

B+-tree; Index; PCM

1. INTRODUCTION

The increasing needs for large energy-efficient main memory call for new types of memories, such as phase change memory (PCM). PCM is byte-addressable and supports random access. Compared with DRAM, PCM is non-volatile and is expected to have higher storage density in the future [1, 2]. However, two problems make it difficult to replace DRAM in current computer systems. First, the write latency of PCM is about 6 to 10 times slower than that of DRAM. Second, PCM cells have limited write endurance [3].

Therefore, a more practical way to utilize PCM in memory architecture is to use both PCM and DRAM to construct hybrid memory architecture [2]. In such hybrid memory architecture, the high density and non-volatility of PCM makes it possible to build non-volatile main-memory databases. However, due to the special properties of PCM, many existing database algorithms such as indexing have to be revised to take advantage of PCM.

In this paper, we focus on the indexing issue in PCM/DRAM-based hybrid memory systems. We aim to improve the traditional B+-tree to make use of PCM and DRAM efficiently. Specifically, we improve the traditional B+-tree in two aspects. First, we use an overflowing mechanism [4, 5] to reduce write operations to PCM, where each leaf node in the B+-tree is allowed to have several overflow nodes to keep newly inserted records when the leaf node is full. Thus, we can reduce the split operations on the index and consequently reduce writes and lengthen the lifetime of PCM. Second, we propose to predict the read/write tendency of index requests, based on which we use different ways to process index requests. Particularly, we use the overflowing scheme for write-intensive requests, but adopt a tree-adjusting operation to remove overflow nodes so as to improve read performance. We conduct experiments to evaluate our proposal and make comparison with the traditional B+-tree and the overflow B+-tree. The results suggest the efficiency of our proposal.

2. CB+-TREE

The structure of the CB+-tree is similar with the B+-tree, except that each leaf node can have a few overflow nodes. A leaf node without overflow nodes is called a tra, while a leaf node containing overflow nodes is called an ovf. All the data stored in leaf nodes of CB+-tree is the same as that in the B+-tree. Nodes of the CB+-tree are ordered, thus binary search is allowed. Every node of the CB+-tree consists of a set of <key, value> pairs and some auxiliary information. The auxiliary information includes num_keys, is_leaf, parent, and brother. Here, num_keys denotes the number of <key, value> pairs, is_leaf indicates whether the node is a leaf node, and parent and brother denote the parent and the brother of the node, respectively.

In the CB+-tree, when an ovf leaf node has a read tendency, we remove it from the overflow chain and make it be a new tra. With this mechanism, we can reduce the number of overflow nodes and therefore improve read performance. As shown in Fig. 1, the leaf nodes LN1, LN2 are tra and LN3 is an ovf. If LN3 is required to be changed into a tra, we remove LN3 from its original overflow chain and change it into a tra. Other ovf in the original overflow chain remain unchanged. Consequently, the overflow chain is changed into two segments, as shown in the right part of Fig. 1.

Figure 1. Structure adjustment of the CB+-tree.

The key issue in the CB+-tree is how to predict the read/write tendency (T) of a leaf node. Formally, T is calculated by (1). Here, \( \text{ratio}_0 \) and \( \text{ratio}_1 \) are total and recent proportion of writes to all accesses on a leaf node, respectively.

\[ T = \text{ratio}_0 \cdot \delta + \text{ratio}_1 \cdot (1 - \delta), \quad s.t. \, \delta \in [0, 1]. \]  

We make use of a sliding window to record \( k \) latest read/write requests for a leaf node, and further get the value of \( \text{ratio}_0 \). Generally, a node is considered to be write-intensive when \( T \) is higher than 0.7, and be read-intensive when \( T \) is lower than 0.3. The read/write tendency prediction scheme works when \( T \) is in 0.3-0.5 for an ovf and in 0.5-0.7 for a tra.

© 2016, Copyright is with the authors. Published in Proc. 19th International Conference on Extending Database Technology (EDBT), March 15-18, 2016 - Bordeaux, France: ISBN 978-3-89318-070-7, on OpenProceedings.org. Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0
We perform the prediction by using the polynomial fit technique according to the recent access information in the sliding window. In addition, we do not trigger the prediction when the T of an ovf drops a little below 0.7 immediately, but wait for some time to collect more information about the access pattern. This strategy is also applied to the change of the parameter tra.

3. EVALUATION
To observe the performance benefits of the proposed CB+-tree, we implemented three algorithms: the traditional B+-tree, the OB+-tree[4] and the CB+-tree, and we run them on a computer with Ubuntu 14.04, a CPU of AMD Athlon II X2, 4GB RAM, and 1 TB Seagate hard disk. In addition, we used DRAM to simulate PCM by artificially increasing write latency.

We used the TPC-C workload to generate the traces in the experiments. When running the TPC-C workload, 10 warehouses and 100 clients are configured. The TPC-C workload contains eight index files built on eight tables, and the size of the tables is approximately 1 GB. We used BenchmarkSQL2 to generate the TPC-C workload running on the open-source PostgreSQL, and collect page requests on the eight tables at the same time. We first performed insertions on the tables to get index insertions. Consequently, we performed 5.4 million insertions requests and the original B+-tree index file is about 135 MB. Then, we prepared a trace containing about 3.8 million index requests including 74.2 % searches, 23.8 % insertions, and 1% deletions.

Note that our index has a little more PCM writes than the OB+-tree. This is due to the read/write-tendency-based adjustment of the index structure. Figure 4 shows the run time of each index, which is normalized according to the run time of the B+-tree, i.e., the run time of the B+-tree is always set to 1. It indicates that the CB+-tree outperforms the B+-tree and the OB+-tree in terms of overall run time. As a result, the CB+-tree is able to balance the read and write costs, yielding a better indexing mechanism for PCM-based memory systems.

4. CONCLUSIONS
PCM has been regarded as a new kind of future memories. In this paper, we proposed an efficient tree indexing approach called CB+-tree that is an improved version of the traditional B+-tree. We used the overflow-node design to reduce writes to PCM, and thus the endurance of PCM can be improved. We also proposed to predict the read/write tendency of index requests, based on which we performed necessary adjustment on the tree index to reduce additional read operations caused by overflow nodes. The comparative experimental results including read/write count and run time, show the efficiency and superiority of our proposal.

5. ACKNOWLEDGMENTS
This work is partially supported by the National Science Foundation of China (61379037 and 61472376).

6. REFERENCES


---

1 http://www.tpc.org/tpcc/
2 http://sourceforge.net/projects/benchmarksql/

---

Figure 2. PCM read counts with different buffer sizes.

Figure 3. PCM write counts with different buffer sizes.

Figure 4. Run Time (normalized w.r.t. B+-tree).