Expert Systems with Applications 42 (2015) 4676–4686

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Optimizing R-tree for flash memory

Peiquan Jin [a,c,*], Xike Xie [b], Na Wang [a], Lihua Yue [a,c]

[a] School of Computer Science and Technology, University of Science and Technology of China, PR China
[b] Department of Computer Science, Aalborg University, Denmark
[c] Key Laboratory of Electromagnetic Space Information, Chinese Academy of Sciences, PR China

ARTICLE INFO

Article history:
Available online 30 January 2015

Keywords:
Spatial index
Flash memory
Buffer management
R-tree

ABSTRACT

R-tree has been widely used in spatial data management and data analysis to improve the performance of spatial data retrieval. However, the original R-tree is designed for magnetic disks, and has poor performance on flash memory, due to the special features of flash memory such as asymmetric read/write speeds (fast read, slow write) and the erase-before-write feature. Particularly, the original updating mechanism of R-tree usually has to update a few interior nodes when inserting an indexing item into or deleting an item from a leaf node, yielding many slow writes to flash memory. With the wide use of flash memory in many location-based fields, e.g., to store moving trajectories in intelligent transportation systems, how to optimize R-tree for flash memory has become a critical issue. In this paper, we propose a novel spatial index named *Flash-Optimized R-tree* that is optimized for flash memory. In particular, we propose to defer the node-splitting operations on R-tree by introducing overflow nodes, which results in an unbalanced tree structure. With this mechanism, we can reduce random writes to flash memory and improve the overall performance of R-tree. In addition, we present a new buffering scheme to efficiently cache the updates to the tree, which can further reduce random writes to flash memory. We conduct extensive experiments on real flash-memory storage devices as well as a flash memory simulation platform to evaluate the performance of our proposal, and the results suggest the efficiency of our proposal with respect to different metrics.

1. Introduction

Recently, flash memory as well as flash-based solid state drives (SSDs) has been widely used in computer systems to accelerate I/O efficiency, owing to its superior features compared with magnetic disks, e.g., smaller size, lighter weight, lower power consumption, shock resistance, lower noise, and faster read performance (Chang & Kuo, 2005; Wu & Kuo, 2006; Xiang, Yue, Liu, & Wei, 2008). Thus, many people have proposed to use flash memory in spatial databases in recent years (Koltsidas & &Viglas, 2011; Lv, Li, Cui, & Chen, 2011; Sarwat, Mokbel, Zhou, & Nath, 2011, 2013; Wu, Chang, & Kuo, 2003). Due to the special features of flash memory that differ from magnetic disks, many data structures and algorithms in spatial data management, such as spatial indices, have to be re-considered.

However, it is not a trivial task to optimize a spatial index for flash memory. Existing spatial indices such as R-tree (Chawla & Sun, 2006; Guttman, 1984; Theodoridis & Sellis, 1996) are mostly proposed for improving search performance. On the other side, they usually have poor update performance. For example, R-tree is a balanced spatial tree index that offers high search performance. However, its balanced structure will incur bottom-up updates on the tree, i.e., insertions and deletions will cause updates on leaf nodes propagating up to the root node. As flash memory has a slow random-write speed, R-tree will achieve poor performance on flash memory (Wu et al., 2003). In addition, as flash memory has a limited erasure count because of its erasure-before-write feature, we have to consider new techniques for R-tree to reduce the erasures to flash memory so as to extend the life cycle of flash memory. As a consequence, optimizing spatial indices for flash memory will lead to substantial re-designs on index structures. In order to reduce random writes to flash memory, we need to revise some key features of traditional spatial indices such as R-tree.

There are also some previous studies focusing on flash-based spatial indexes (Lv et al., 2011; Sarwat et al., 2011, 2013; Wu et al., 2003). Most of them are based on R-tree and employ a lazy updating policy to reduce the costly random writes and erasure operations on flash memory. According to the lazy updating policy, all updates to leaf nodes are cached in main memory, and the node with the most updates is written to flash memory in case of buffer

* Corresponding author at: School of Computer Science and Technology of China, University of Science and Technology of China, Jinzhai Road 96, Hefei 230027, China. Tel.: +86 139 5516 2813.

E-mail addresses: jpq@ustc.edu.cn (P. Jin), xkxie@cs.aau.dk (X. Xie), wangna@mail.ustc.edu.cn (N. Wang), llyue@ustc.edu.cn (L. Yue).

http://dx.doi.org/10.1016/j.eswa.2015.01.011
0957-4174/

replacement. The advantage of the lazy updating mechanism is that it can combine several writes to a leaf node into one write and therefore the overall writes to R-tree can be reduced. However, although we can merge a few writes into one operation, the writes to leaf nodes will still impose node-splitting operations on the tree, which will probably introduce many writes to flash memory. Another problem in existing work is that the buffer replacement algorithms they proposed only consider the update count of the cached nodes and always choose the nodes with the most updates as the victim. This will result in poor performance if the evicted node is to be visited soon, because the temporal locality of buffer references indicates that a recently updated page is much likely to be re-referenced soon.

This paper presents the first optimization on R-tree for reducing node-splitting operations on the index. The motivation of the paper is to postpone node-splitting operations by introducing overflow nodes to R-tree and making R-tree be an unbalanced tree. In addition, we propose a new buffering scheme to cache updates to R-tree, which can further reduce random writes to flash memory. In summary, we make the following contributions in this paper:

(1) We propose a flash-optimized spatial tree index called *Flash-Optimized R-tree* (FOR-tree). FOR-tree is the first optimization on R-tree that considers reducing node-splitting operations by using overflow nodes. FOR-tree is designed as an unbalanced tree but it can reduce random writes and further erasures to flash memory. We also propose a merge-back algorithm for overflow nodes, which can efficiently reduce the read overhead on FOR-tree.
(2) We design new search and update algorithms to adapt the unbalanced structure of FOR-tree. Compared with previous work on flash-aware R-trees, our algorithms can reduce more random writes on flash memory, especially for skewed access patterns.
(3) We propose a new buffering scheme to maintain node updates in a write buffer. The new scheme uses both the update count and the cold/warm attribute of the buffered nodes to select the victim during the buffer replacement process. Compared with previous update-count-based algorithms, our proposal introduces fewer random writes on flash memory.
(4) We conduct extensive experiments on both real SSDs and a flash memory simulation framework, and run one million transactions to evaluate the performance of FOR-tree. The results show that FOR-tree outperforms its competitors considering various metrics such as write count, erasure count, and run time.

The rest of the paper is organized as follows. In Section 2, we review related work. In Section 3, we propose the general design of FOR-tree. In Section 4, we describe the operations on FOR-tree. Section 5 explains the buffering scheme of updates. Section 6 covers the experimental results, and finally Section 7 concludes the paper and discusses the future work.

2. Related work

Due to the characteristics that random writes are much slower than sequential writes and random reads on flash memory, it is common to reduce the number of random writes so as to obtain high update performance. Some flash-aware indices sacrifice sequential writes and random reads to avoid random write operations, such as write-optimized B-tree (Graefe, 2004) and CHC-Tree (Byun & Hur, 2009).

Generally, the update performance on flash memory can be improved by buffering updates in memory and flushing them to flash memory in batches. Wu et al. proposed BFTL (Wu, Chang, & Kuo, 2004) and RFTL (Wu et al., 2003) respectively for B-Tree and R-tree that handle fine-grained updates in a layer over FTL for flash-memory-based disks. They allowed modification entries of different nodes to be stored sequentially in memory, and then packaged them into sectors to write to flash memory when in-memory storage is full. As the number of updates delayed is in proportion to memory size, Kang, Jung, Kang, and Kim (2007) proposed μ-Tree to put all the nodes along the path from the root to the leaf together into a single NAND flash memory page for raw flash memory at the absence of FTL. However, it cannot adapt to large amount of data because it only used small-sized nodes in the index structure. Agrawal, Ganesan, Sitaraman, Diao, and Singh (2009) proposed LA-Tree that used to buffer updates in flash resident cascaded buffers which are corresponded to each two-level sub-tree, and it consequently reduced the traditional use of memory for caching. LA-Tree focused on raw, small-capacity and byte addressable flash memory. Li, He, Luo, and Yi (2009), Li, Jin, Su, Cui, and Yue (2009), Li, He, Yang, Luo, and Yi (2010) proposed FD-Tree for off-the-shelf large flash SSDs, which improved update performance by allowing main memory resident head tree and organizing elements of remaining levels in an ascending order, while improved search performance by storing fence pointer in each level. However, FD-Tree needs to reorganize all its entries when the merge operation generates a new level.

LCR-tree (Lv et al., 2011) as an efficient spatial index for SSDs was designed to combine newly arrival log with origin ones on the same node, which renders great decrement of random writes with at most one additional read for each node access. However, it needs to read all origin logs and rewrite them for each modification. Sarwat et al. (2011, 2013) proposed FAST-Rtree that used an external upper-layer framework which encapsulates the original algorithms. In FAST-Rtree, updates are buffered in main memory and hashed by node identifier so as to be flushed to flash memory in batches. However, it has not considered reducing node-splitting operations. To the best of our knowledge, FAST-Rtree is the most-recently spatial index proposed for flash memory. Therefore, we will mainly compare our proposal with this index in our experiments.

Srinivasan (1991) presented an adaptive overflow technique to defer node splitting in B-Trees to improve the storage utilization, however it did not consider merging back the overflow node until it was full. (Manousaka & Manolopoulos, 2000) proposed lazy parent split to achieve better concurrency while it did not reduce the number of random write operations. Bozanis, Nanopoulos, and Manolopoulos (2003) proposed a logarithmic decomposable spatial index method named as LR-tree based on R*-tree (Beckmann, Kriegel, Schneider, & Seeger, 1990) to avoid performance tuning with forced reinsertion, each component sub-structure called block is organized as a weak R-tree. Whenever an insertion operation must be served, a set of blocks are chosen to first destroy and then reconstruct using bulk-loading. However, the invalid space in nodes of sub-structures due to the deletion-only principle and the additional bulk-loading cost in LR-tree reconstruction phrase cannot be neglected. Koltsidas and Viglas (2011) presented desiderata for improved I/O performance of spatial tree index over flash memory and gave a brief analysis of the unbalanced structure of spatial tree index. In this paper, we also take into account the unbalanced feature of spatial tree indexes.

Although a few other spatial indices have been proposed for accelerating spatial queries, such as location-based grid index (Park, 2014), there are few pieces of work focusing on optimizing them for flash memory. This is mainly due to the generality and high efficiency of R-tree compared with other spatial indices.