Storing Trees on Disk Drives

Medha Bhadkamkar Fernando Farfan Vagelis Hristidis Raju Rangaswami
School of Computing and Information Sciences
Florida International University
{medha,ffarf001,vagelis,raju}@cis.fiu.edu

1. INTRODUCTION

Tree-structured data are abundant today, ranging from Bioinformatics suffix-tree alignments, to multi-resolution video, to directory-file hierarchies, to XML. The storage techniques employed by systems that manage tree-structured data greatly affect their performance. Current approaches either map the tree data to an underlying relational database system, or use the abstraction provided by a general-purpose object storage manager, or simply use flat files. These storage schemes, however, ignore the tree structure of the data as well as the characteristics of disk drives. Relational databases are structured tables and flat files are unstructured. On the other hand, disk drives store information in circular tracks that are accessed with mechanical seek and rotational overhead. The performance of disk drives greatly depends on the I/O access pattern (orders of magnitude difference between sequential and random access times). To the best of our knowledge, there exists no data layout strategy that accounts for the structural mismatch between tree-structured data and disk drive storage.

We propose a new storage technique, *tree-structured placement*, that explicitly accounts for the mismatch between tree-structured data and disk drive characteristics, so that common navigation operations (parent-to-child and node-to-next-sibling) are efficient. This technique uses the recently proposed idea of semi-sequential disk access [2] to place the tree structure. It also presents optimizations that reduce the on-disk space fragmentation and average random seek-times. Experimental evaluation using the DiskSim disk simulator [1] suggests as much as 80% reduction in query IO times compared to the default sequential layout of tree-structured data.

2. TREE-STRUCTURED PLACEMENT

Using tree-structured placement, tree nodes are placed on the disk starting from the outermost available track. In particular, we first place the root node v of the tree data on the block with the smallest logical-block-number (LBN), on the outermost available track of the disk. Second, we place its children sequentially on the next *free* track such that accessing the first child u of v after accessing v results in a *semi-sequential access* Further, accessing a non-first child from a parent node involves a semi-sequential access to reach the first child and a short rotational-delay based on the child index. The children of the first-child of the root node are then placed next using a similar approach, followed by those of the second-child and so on. The remaining nodes of the data tree are placed following a similar approach.

The above placement strategy, though simple, incurs significant fragmentation in disk space as well as large average random seek-times as a consequence. The *optimized tree-structured placement* strategy allows placement of the child nodes in non-free tracks, thereby reducing fragmentation. Second, it allows the first-child node anywhere in a rotationally-optimal track-region[1] relative to the parent rather than requiring it to be place at the rotationally-optimal block. This allows child nodes to be placed closer to the parent node, thereby reducing seek-times. However, the trade-off here is slightly increased average rotational delays. Third, it groups tree nodes into supernodes that map to disk blocks, further reducing the fragmentation within disk blocks. These optimizations make the tree-structured placement significantly superior to the default sequential approach as detailed below.

3. EVALUATION

We used tree-structured XML data (benchmark navigational queries (XPath) and benchmark XML databases [3]) to evaluate our approach. The baseline strategy that we compare our approach against is sequential layout (referred to as *default*) of the XML file on the disk, as provided by general-purpose filesystems. Figure 1 shows the average total IO times for 8 benchmark XPath queries for 4 different disk drive models simulated using DiskSim [1]. Tree-preserving tree-structured placement (*TP-TS*) groups the XML nodes to supernodes in a way that the tree structure is preserved, whereas Sequential tree-structured placement (*Seq-TS*) uses the same grouping as default and fits as many nodes as possible into a supernode.

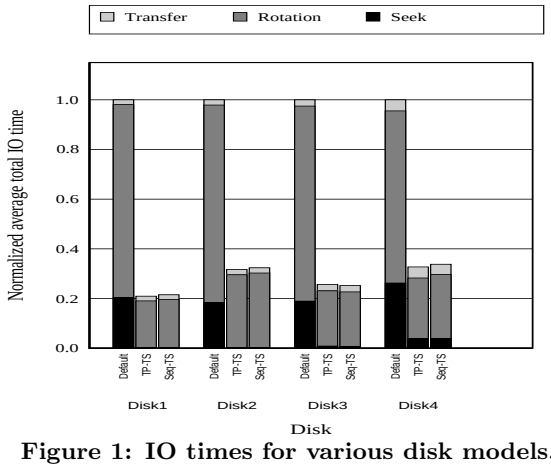

Figure 1: IO times for various disk models.

4. REFERENCES

[1] J. Bucy, G. Ganger, and Contributors. The DiskSim Simulation Environment Version 3.0 Reference Manual. *Carnegie Mellon University Technical Report CMU-CS-03-102*, January 2003.

[2] J. Schindler, S. W. Schlosser, M. Shao, A. Ailamaki, and G. R. Ganger. Atropos: A Disk Array Volume Manager for Orchestrated Use of Disks. *Proceedings of the USENIX Conference on File and Storage Technologies*, March 2004.

[3] A. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, and R. Busse. XMark: A Benchmark for XML Data Management. *VLDB*, 2002.

[1] defined as a fixed-length sequence of blocks starting from the rotationally-optimal block