Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Information Processing Letters 86 (2003) 197–202

Information Processing Letters

www.elsevier.com/locate/ipl

Data replication in static tree structures ☆,☆☆

Susanne E. Hambrusch *, Chuan-Ming Liu

Department of Computer Sciences, Purdue University, West Lafayette, IN 47907, USA

Received 3 June 2002

Communicated by F. Dehne

Keywords: Algorithms; Blocknumber; Data replication; Secondary storage; Static tree structures

1. Introduction

Replication of data can lead to better performance due to increased availability of data. This paper explores the use of data replication for large, static data sets organized in a tree structure. In static environments, data replication does not have to deal with problems arising from maintaining consistency among copies [1,10,12]. Static data sets arise, for example, when updates on the data happen at fixed times and in bulk, and queries are not present at the time of updating. We show that in static tree structures data replication is effective in reducing the amount of I/O needed measured in terms of the blocknumber. Our techniques determine what data to replicate, how to control the amount of replication, and how to ensure good block utilization. Our results improve previously known bounds.

Let T be an N-node rooted tree with r as the root. When the data associated with the nodes of T is too large to store in main memory, nodes of T are mapped to blocks of size at most B. Blocks are stored externally and accessing one block is considered one I/O operation. In this paper we assume that one block can hold the data of B nodes. Hence, at least $\lceil N/B \rceil$ blocks are stored externally. Two metrics used to measure the quality of the generated blocks are (i) the number of nodes assigned to blocks and (ii) the blocknumber. When at most one block is assigned fewer than B nodes we refer to the mapping as a *complete mapping*. Blocks containing fewer than B nodes are undesirable since they underutilize resources. The blocknumber measures external access during a search. Consider a path P from root r to a leaf l in T. The *blocknumber* is the maximum number of edges (u, v) on path P for which u and v are in different blocks.

In this paper we describe complete mappings from the nodes of T to blocks when nodes can be replicated, the amount of replication is controlled, and the blocknumber is optimized. A node of T can be mapped to more than one block and is thus available in different blocks. A mapping of tree T has a *total replication factor* τ if the the number of nodes in all blocks is bounded by τN, $\tau \geqslant 1$. We show that any tree T of

☆ Work supported in part by the NSF under grants 9988339-CCR and 0010044-CCR.

☆☆ A preliminary version of this paper appeared in the *Proceedings 9th CIKM*, 2000.

* Corresponding author.
E-mail addresses: seh@cs.purdue.edu (S.E. Hambrusch), liucm@cs.purdue.edu (C.-M. Liu).

0020-0190/03/$ – see front matter
doi:10.1016/S0020-0190(02)00503-3

height h has a complete mapping with a total replication factor of $\tau = \frac{3}{2}$ and achieving a blocknumber of at most $\lceil (h - \log_d B)/B \rceil + 2$. Hence, in an asymptotic sense, by using at most 50% more space, we can achieve the best possible blocknumber. We also extend our approach to smaller replication factors.

Mapping search trees to external storage minimizing the cost measures defined above has been considered in [2,3,7,8,11]. The use of data replication in static environments related to our work has been studied in [4,5,7,9]. In particular, Nodine et al. [7] describe a mapping of a complete binary tree of height h for $\tau = 2$ achieving a blocknumber of $\lceil h/\log B \rceil$. Hutchinson et al. [5] present a mapping using $\tau = 2 + \varepsilon$ and achieving a blocknumber of $\varepsilon h/(2-\varepsilon)B$. Our results improve these bounds. We also extend the bounds of Diwan et al. [2] by showing that the optimal blocknumber increases by at most 1 for complete mappings when no replication is used. For a search following a path towards the root, the blocknumber is a true measure of the number of external blocks accessed. This holds for our as well as the cited results. For searches towards the leaves, mappings, including ours, can have limitations. Additional information would need to be added to make the blocknumber of the mapping correspond to the blocknumber of the actual search. We point out that this is not the focus of our work. We show that there exist mappings achieving a good blocknumber and on how replication impacts the blocknumber.

2. Preliminaries

In this section we review results on blocknumbers when nodes of the tree cannot be replicated. Observe that in this case a mapping corresponds to a partition of the nodes. Assume tree T has height h. Let d be the maximum number of children of a node in T, $d \geqslant 2$. Let bl^* be the optimum blocknumber in a not necessarily complete mapping of T.

For any tree T having height h we trivially have $bl^* \geqslant \lceil h/B \rceil$. Using known techniques, one can show that for any N-node tree there exists a mapping achieving a blocknumber of at most $\lceil h/\log_d B \rceil$. For complete d-ary trees this is the best possible blocknumber; i.e.,

$$bl^* = \lceil h/\log_d B \rceil.$$

Proofs can be found in [6].

In [2], Diwan et al. present a linear time algorithm generating blocks containing between $B/2$ and B nodes and achieving minimum blocknumber (without nodes replication). We sketch a "ripple-down" technique (similar to the one used in [3]), that generates a complete mapping P' having blocknumber at most $bl + 1$ from any mapping P having blocknumber bl. The ripple-down process is used by the algorithm described in Section 3.

The complete mapping P' is obtained from P in two phases. First, the blocks of P are considered level-by-level starting with the block containing root r. If a block, say block X, contains fewer than B nodes and not all nodes correspond to leaves of T, children of leaves in X are reassigned to block X. Nodes are reassigned to block X until X either contains B nodes or all leaves of X correspond to leaves in T. This reassignment of nodes does not increase the blocknumber.

Assume the reassignment generates m blocks containing fewer than B nodes. Let $B_1, \ldots, B_m$ be these blocks and $n_1 \leqslant n_2 \leqslant \cdots \leqslant n_m < B$ be the number of nodes in these blocks. The second phase generates a complete mapping by reassigning subtrees. From the actions taken in the first phase it follows that every leaf in a block B_i is a leaf of T. Assume the nodes in blocks $B_1, \ldots, B_{i-1}$ have already been assigned to their final blocks in complete mapping P'. Block B_i may have been changed by the actions already taken. We next determine the index j such that $n_i + \cdots + n_{i+j-1} \leqslant B$ and $n_i + \cdots + n_{i+j} > B$. In the case of equality, $B_i, \ldots, B_{i+j}$ form one block in P'. Otherwise, we assign the nodes in blocks $B_{i+1}, \ldots, B_{i+j-1}$ to block B_i. To make block B_i complete, it receives from block B_{i+j} the number of nodes still needed to be complete. The nodes assigned to B_i form either a single subtree in B_{i+j} or their removal leaves one subtree in B_{i+j}. (In case B_{i+j} contains more than one tree, a traversal of one tree in B_{i+j} generates the desired result.) We then continue with block B_{i+j+1}.

After the reassignment, at most one block contains fewer than B nodes and we thus have a complete mapping P'. The blocknumber increased by at most one during the reassignment phase. We can thus conclude the following:

Theorem 1. *Let T be an arbitrary tree and let bl^* be the optimum blocknumber in a non-complete mapping*

of T. Then, there exists a complete mapping for T achieving a blocknumber of at most $bl^* + 1$.

3. Mappings with a total replication bound

In this section we first describe an algorithm which generates a complete mapping having a blocknumber of at most $\lceil (h - \log_d B)/B \rceil + 2$ and using at most $\frac{3}{2}N$ nodes in all blocks. The mapping can be generated in $O(N)$ time. This compares favorably to the blocknumber of $\Theta(h/\log_d B)$ which is the best possible when $\tau = 1$ (i.e., no node replication). Our algorithm consists of three phases: preprocessing, path creating, and balancing. We discuss each one of the three phases in detail.

Phase 1: *Preprocessing*

For every node v in T determine $desc(v)$, the number of descendents in the subtree rooted at v. Let $p(v)$ be the parent of node v in T. A node v is marked as a *subtree leaf* if $desc(v) \leqslant 2B$ and $desc(p(v)) > 2B$. A subtree leaf represents a subtree of T containing at most $2B$ nodes. A subtree leaf v and its descendents form one block. The descendents of a subtree leaf v are removed from T and the resulting new tree is used in Phase 2.

Fig. 1(a) shows a tree T with $B = 6$ after the preprocessing phase. A subtree leaf corresponds to at most 12 nodes and the parent of a subtree leaf is the root of a subtree containing more than 12 nodes.

Phase 2: *Path-creating*

Let T' be the tree generated by Phase 1. The path-creating phase performs a bottom-up traversal of T' in which subtrees of height B are identified and removed from T'. The subtrees are used to generate paths which are assigned to blocks. Subtrees containing root r may generate blocks with fewer than B nodes.

Let H be a subtree of T' with root r_H having height B. In the first subtrees identified, the leaves of H correspond to nodes of T marked as subtree leaves. In the later steps, leaves of H will either be subtree leaves or path leaves (path leaves will be defined shortly). Node e of H is an *end-node* if all children of e are either subtree leaves or path leaves. For every end-node e in H, we create a block containing the nodes on the path from root r_H to e. We refer to r_H as the start-node and to e as the end-node of the path. After all paths have been generated for H, subtree H—with the exception of root r_H—is removed from T'. Node r_H is marked as a *path leaf*.

When applying the path-creating phase to the tree shown in Fig. 1(a), node k is identified as the root of a subtree H of height $B = 6$. Node p is an end-node

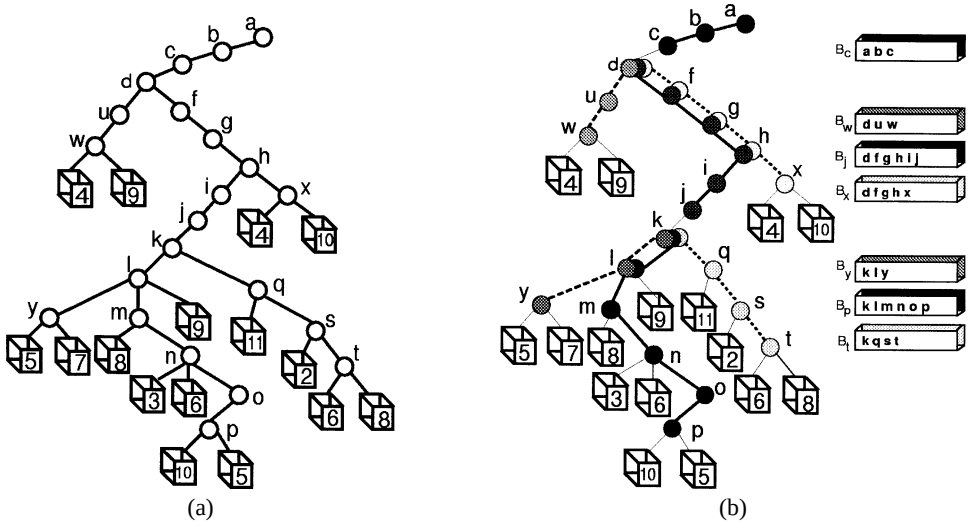

Fig. 1. Illustrating Phases 1 and 2 for $B = 6$; cubes represent subtree leaves; the integer on the cube represents the number of nodes in the subtree. (a) Tree T after preprocessing, (b) after path-creating. Blocks created in (b) are indicated in the tree and are also shown explicitly on the right.

in H and the nodes on the path from k to p form one complete block, block B_p. Subtree H contains two other end-nodes, nodes y and t. They induce two more blocks, B_y and B_t, as shown in Fig. 1(b). Block B_y contains 3 nodes and has y as the end-node and block B_t contains four nodes and has t as the end-node. Phase 2 thus generates three copies of node k. Four more blocks, namely B_j, B_x, B_w, and B_c, are generated in Phase 2.

Phase 3: *Balancing*

After the path-creating phase, every block contains between 1 and $2B$ nodes of T representing either a path or a subtree. A block assigned a path contains at most B nodes and a block assigned a subtree contains at most $2B$ nodes. The balancing phase generates a complete mapping by reassigning nodes. The reassignments increase the blocknumber by at most 2. The balancing phase proceeds top–down. It starts with the blocks containing a copy of the root of T. Assume the current block contains a path P of length $< B$. The end-node e of path P is incident to nodes which are either in paths or in subtrees.

Case 1: *End-node e has at least one child which is the start-node of a path in a block.* In this case there exists at least one child of e which is in a block containing a path of length exactly B. Choose one such child, say v, and let P_v be the path of length B containing v as the start-node. In order to make the block containing path P complete, reassign $B - |P|$ nodes from path P_v to the block containing path P. Subsequent steps will make the block containing the shortened path P_v complete.

Fig. 2 illustrates this step. As indicated in Fig. 1(b), block B_c initially contains three nodes, including root a and c as the end-node. Block B_j contains 6 nodes and has a child of c as the start-node of its path. Block B_c is made complete by taking the first three nodes of the path in B_j. Subsequently, block B_j contains three nodes, with h being the new start-node. Block B_j is made complete by taking three nodes of block B_p.

Case 2: *All children of end-node e are subtree leaves.* When e is incident to only subtree leaves, we reassign, in a greedy fashion, as many nodes as necessary to create a block of size B. If possible, we use subtrees in its entirety. If this is not possible, we include the needed number of nodes in a level-by-level manner.

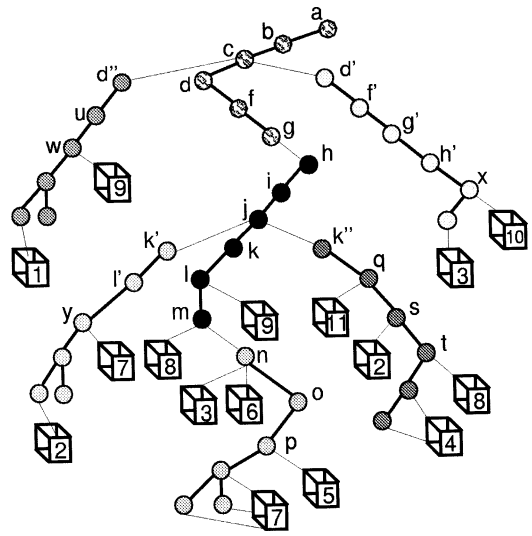

Fig. 2. Illustrating phase 3 for $B = 6$.

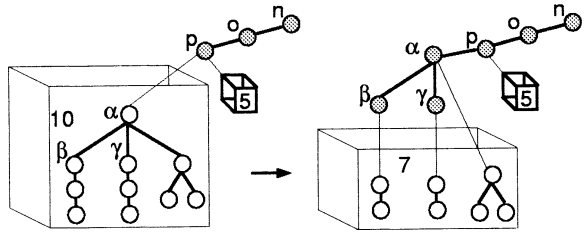

Fig. 3. Making block B_p complete: nodes α, β, and γ are assigned to B_p.

We illustrate this case using block B_p from Fig. 1(b). End-node p has two children, one representing a subtree of size 10 and one of size 5. Block B_p becomes complete by taking three nodes from subtrees rooted at the children of p. In the solution shown in Fig. 3, three nodes from the subtree of size 10 are assigned to block B_p. This assignment partitions the seven remaining nodes into three subtrees, as indicated in Fig. 3. The light edges in Fig. 2 indicate adjacency relations to nodes in other blocks.

After all blocks containing paths have been handled, every block which originally contained a path is complete and no change in the blocknumber occurred. Blocks containing subtrees contain between 1 and $2B$ nodes. We first consider the blocks containing more than B nodes. For each such block, we iden-

tify the exactly B nodes by either choosing subtrees or choosing nodes from one subtree in a breadth-first manner. These B nodes form a new block. Creating complete blocks this way increases the blocknumber of the mapping by at most 1. Blocks containing subtrees with fewer than B nodes are handled using the ripple-down process described in Section 2 Resulting reassignments can increase the blocknumber by at most 1.

Using standard tree operations, the algorithm described has a straightforwards $O(N)$ time internal memory implementation. We refer to [6] for details.

Theorem 2. *The nodes of an arbitrary N-node tree T can be mapped to complete blocks using at total replication factor $\tau < \frac{3}{2}$ and achieving a blocknumber of at most $\lceil (h - \log_d B)/B \rceil + 2$.*

Proof. Consider first the blocknumber of the mapping. The preprocessing phase identifies subtrees of height at least $\log_d B$. Once these subtrees are removed from T, a path from the root to a subtree leaf encounters at most $\lceil (h - \log_d B)/B \rceil$ blocks. Since the balancing phase increases the final blocknumber by at most 2, the claimed bound follows. In an asymptotic sense, the blocknumber achieved by the mapping is optimal.

We next show $\tau < \frac{3}{2}$. The path-creating phase starts operating on a tree T' obtained from T by forming subtree leaves. Recall that a subtree leaf is the root of a subtree in T of size at most $2B$ (and the parent of v is the root of a subtree of size $> 2B$). The bound on the total space needed by the mapping uses a "charging argument". We show that every node of tree T receives at most two charges and that the total number of charges is bounded by $\frac{3}{2}N$.

Let l be a subtree leaf. Every node of T in the subtree rooted at l receives one charge. Let H be a subtree of T' selected in the path-creating phase. Every subtree selected (except possibly the last subtree containing the root of T) has height B. Let P be a path from the root of H to an end-node e. The length of path P is at most B. We do not charge the nodes on path P. (Charging these nodes would give a node a charge equal to the number of times the node is replicated.) Consider first the situation when all the children of end-node e are subtree leaves. The total number of nodes in the subtrees rooted at the subtree leaves is at least $2B$. We give $|P| \leqslant B$ of these nodes one additional charge. Observe that every node in such a subtree has already been charged once and thus at most half of them end up with two charges.

When end-node e contains a child which is a path leaf l, the charging is done as follows. Path P from the root of H to end-node e has length at most B. Node l is the start-node of a path P' of length B in a subtree handled earlier in the path-creating phase. None of the nodes on path P' have yet been charged. We now assign a charge of 1 to $|P|$ nodes on path P'. Later steps assign no further charge to the nodes on path P'.

In summary, if all children of a node v are subtree leaves, then at most B nodes in the subtrees rooted at v receive two charges and at least B of these nodes receive only one charge. All other nodes of tree T receive at most one charge. Thus, at most half of the N nodes of T receive two charge and the total number of charges made is at most $\frac{3}{2}N$. □

Since a total replication factor of $\frac{3}{2}$ results in an asymptotically optimal blocknumber, we do not consider larger replications. Our approach can be generalized to smaller replication factors and a tradeoff between total replication and blocknumber can be established. For arbitrary τ, $1 \leqslant \tau < \frac{3}{2}$, the preprocessing phase identifies the subtree leaves as the nodes v which have

$$desc(v) \leqslant \frac{1}{\tau - 1}B \quad \text{and} \quad desc\big(p(v)\big) > \frac{1}{\tau - 1}B.$$

The path-creating phase generates a mapping in which a block contains between 1 and $B/(\tau - 1)$ nodes and the blocknumber is at most $\lceil (h - \log_d \frac{1}{\tau - 1}B)/B \rceil$. The balancing phase increases the blocknumber by at most $\lceil \frac{1}{\tau - 1} \rceil$. In summary, using $O(\tau N)$ space, we can generate a complete mapping of an arbitrary tree achieving a blocknumber of at most

$$\left\lceil \left(h - \log_d \frac{1}{\tau - 1}B \right) \Big/ B \right\rceil + \left\lceil \frac{1}{\tau - 1} \right\rceil.$$

4. Conclusion

We presented complete mappings of static tree structures to blocks of size B under the assumption that nodes can be replicated. The quality of a mapping is measured by the blocknumber and the amount of

replication. Our results showed that by allowing 50% more space, we can generate a complete mapping achieving a blocknumber proportional to the optimal one of $\lceil h/B \rceil$. Bounding the total replication allows replication decisions to be made on the basis of which replications are the most useful ones, while keeping the total space used under control.

References

[1] Y. Breitbart, H.F. Korth, Replication and consistency in a distributed environment, J. Comput. System Sci. 59 (1999) 29–69.

[2] A.A. Diwan, S. Rane, S. Seshadri, S. Sudarshan, Clustering techniques for minimizing external path length, in: Proc. 22nd Internat. Conf. on Very Large Data Bases, 1996, pp. 342–353.

[3] J. Gil, A. Itai, How to pack trees, J. Algorithms 32 (1999) 108–132.

[4] J.M. Hellerstein, E. Koutsoupias, C.H. Papadimitriou, On the analysis of indexing schemes, in: Proc. 16th ACM Symp. Principles of Database Systems, 1997, pp. 249–256.

[5] D. Hutchinson, A. Maheshwari, N. Zeh, An external memory data structure for shortest path queries (extended abstract), in: Proc. 5th Annual Internat. Conf. Computing and Combinatorics, COCOON, Springer, Berlin, 1999, pp. 51–60.

[6] C.-M. Liu, Broadcasting and blocking large data sets with an index tree, PhD Thesis, Purdue University, West Lafayette, IN, May 2002.

[7] M.H. Nodine, M.T. Goodrich, J.S. Vitter, Blocking for external graph searching, Algorithmica 16 (2) (1996) 181–214.

[8] S. Ramaswamy, S. Subramanian, Path caching: A technique for optimal external searching, in: Proc. 13th ACM Symp. Principles of Database Systems, 1994, pp. 25–35.

[9] S. Subramanian, S. Ramaswamy, The P-range tree: A new data structure for range searching in secondary memory, in: Proc. 6th Annual SODA, 1995, pp. 378–387.

[10] P. Triantafillou, F. Xiao, Supporting partial data accesses to replicated data, in: Proc. 10th Internat. Conf. Data Engineering, February 1994, pp. 32–42.

[11] J.S. Vitter, External memory algorithms, in: Proc. 17th ACM Symp. Principles of Database Systems, 1998, pp. 119–128.

[12] O. Wolfson, S. Jajodia, Y. Huang, An adaptive data replication algorithm, ACM Trans. Database Systems 22 (2) (1997) 255–314.