# Data Replication for External Searching in Static Tree Structures

Susanne E. Hambrusch
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
seh@cs.purdue.edu

Chuan-Ming Liu
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
liucm@cs.purdue.edu

## ABSTRACT

This paper explores the use of data replication to improve external searching in static tree structures. We present general and efficient mappings from the nodes of a tree $T$ to blocks of size $B$ when nodes of $T$ can be replicated. The amount of replication is controlled and block utilization and blocknumber are optimized. We consider total node replication (measuring the total space used) and individual node replication (measuring the replication of indivdual nodes). For an arbitrary tree $T$ of size $N$ and height $h$, we show that by using at most $\frac{3}{2}N$ space one can achieve a blocknumber proportional to the optimal blocknumber of $\lceil h/B \rceil$. We show that when every node can be replicated only a constant number of times, no significant reduction in the blocknumber may be possible. Our work also shows that generating mappings in which all but one block contain exactly $B$ nodes increases the blocknumber by at most 2.

## 1. INTRODUCTION

Data replication allows better performance with increased availability of data. In static environments, data replication does not have to deal with maintaining consistency among multiple copies [1, 3, 6, 11, 13, 14]. Static environments dealing with large data sets arise frequently. They include environments in which updates happen at certain, chosen points in time and in bulk, and queries are not present at the time of updating. In this paper we show how to effectively use data replication to improve external searching in static tree search structures. Trees are fundamental data structures for efficient query processing [9, 12] for which mappings to external storage have been developed [2, 4, 7, 8]. The use of data replication in static environments has been studied in [5, 7, 10]. We present general and efficient techniques for mapping trees to external storage. Our techniques determine what data to replicate, how to control the amount of replication, and how to ensure good block utilization.

Let $T$ be a rooted tree consisting of $N$ nodes. When the data associated with the nodes of $T$ cannot be replicated and $T$ is too large to store in main memory, $T$ is partitioned into blocks of size at most $B$. We assume that one block can hold the data of $B$ nodes. We thus generate at least $\lceil N/B \rceil$ blocks to be assigned to external storage. Two metrics commonly used to measure the quality of the generated blocks are the (i) number of nodes assigned to blocks and (ii) the blocknumber. Blocks containing fewer than $B$ nodes are undesirable since they underutilize resources which can result in increased access to the external devices. 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 mappings from the nodes of $T$ to blocks when nodes can be replicated, the amount of replication is controlled, and block utilization and blocknumber are optimized. A node of $T$ can be mapped to more than one block and is thus available in different blocks. We consider two forms of node replication: total node replication and individual node replication. For total node replication we measure the total number of nodes. A mapping of tree $T$ has a *total replication factor* $\tau$ if the the number of nodes in all blocks is bounded by $\tau N$, $\tau \geq 1$. A mapping has an *individual node replication* $\rho$, if every node is replicated at most $\rho$ times, $\rho \geq 1$. Clearly, if a mapping has node replication $\rho$, it also has a total replication of at most $\rho$. For both forms of replication we generate mappings which minimize the blocknumber and achieve a good block utilization. Block utilization is captured by distinguishing between complete and non-complete mappings. When at most one block is assigned fewer than $B$ nodes we refer to the mapping as a *complete mapping*.

In Section 2 we establish bounds on block utilization and blocknumber for the case when no replication is allowed. In Section 3 we show that any tree $T$ of height $h$ has a complete mapping to blocks of size $B$ with a total replication factor of $\frac{3}{2}$ and achieving a blocknumber of $O(h/B)$. Hence, in an asymptotic sense, by using at most 50% more space, we can achieve the best possible blocknumber. The number of times a node is replicated varies in this mapping. Some nodes experience no replication and are thus assigned to only one block; in the extreme case, a node can be replicated $2^B$

times. We also show that extending our solution to smaller replication factors results in a tradeoff between the achieved blocknumber and replication factor. In Section 4 we describe mappings not exceeding a given node replication factor $\rho$. For a constant node replication factor we show that there exist trees for which no significant reduction in the blocknumber is possible. Using a technique leading to optimal mappings with fixed node replication for certain classes of trees, we describe an algorithm achieving fixed node replication for arbitrary trees. Section 5 concludes our paper.

## 2. MINIMIZING THE BLOCKNUMBER WITHOUT REPLICATION

In this section we present relevant results for the generation of blocks when nodes of the search tree cannot be replicated. Observe that in this case a mapping corresponds to a partition of the nodes. The developed bounds are useful when comparing the performance gains in mappings with node replication. Assume tree $T$ has height $h$ and let $d$ be the maximum number of children of a node in $T$, $d \geq 2$. Let $bl^*$ be the optimum blocknumber in a non-complete mapping of $T$. We first show that for any tree $T$ having height $h$ we have

$$\lceil \frac{h}{B} \rceil \leq bl^* \leq \lceil \frac{h}{\log_d B} \rceil.$$

Then, we describe a general technique, similar to [4], for generating a complete mapping from a non-complete mapping (i.e., from a mapping in which there is no restriction on how full blocks have to be). The problem of minimizing the blocknumber in a tree without replication has been considered in [2]. Diwan et al. present a linear time algorithm generating blocks containing between $B/2$ and $B$ nodes and achieving minimum blocknumber. We will show that, by applying our technique, generating the complete mapping increases the initial blocknumber of $bl^*$ by at most 1.

For any tree $T$ having height $h$ we trivially have $bl^* \geq \lceil \frac{h}{B} \rceil$. Before we show the upper bound, we first prove that for $d = 2$ there exists a tree with $bl^* = \Theta(\frac{h}{\log_2 B})$. The generalization of this bound for arbitrary $d$ is straightforward.

Lemma 1. *An $N$-node complete binary tree has $bl^* = \lceil \frac{h}{\lfloor \log_2 B \rfloor} \rceil$.*

**Proof:** Let $T_2$ be the $N$-node complete binary tree. To show that the claimed blocknumber can be achieved, identify in $T_2$ complete subtrees containing exactly $B$ nodes. The first such subtree contains the root of $T_2$ and the other subtrees are identified in a breadth-first manner. Every subtree generated forms a block. Once no more subtrees of size $B$ can be found, each remaining subtree forms a block containing fewer than $B$ nodes. Since no complete mapping is required, no further optimization is needed. We refer to it as mapping $P_2$. The blocknumber of $P_2$ is $\lceil \frac{h}{\lfloor \log_2 B \rfloor} \rceil$.

We next show that the blocknumber achieved by mapping $P_2$ is optimal. Assume there exists another mapping $P$ achieving a smaller blocknumber. Starting with the root of $T_2$ and looking level-by-level, locate the first block in $P$ which is different from a block in $P_2$. Such a block must exist. In this block of mapping $P$ we identify a node $v$ which is a leaf in the subtree assigned to the block, but which is not a leaf in the block in $P_2$. Choose $v$ such that the difference in the levels is a maximum. When a search proceeds from node $v$ to a child of $v$, a new block is needed. Continue with a block containing a child of $v$ and repeat the above argument. The number of blocks requested by partition $P$ will be equal or larger than the number of blocks requested for mapping $P_2$, contradicting our assumption that $P$ has a smaller blocknumber. □

While for a complete tree a blocknumber of $\lceil \frac{h}{\log_d B} \rceil$ is the best possible, we can do better for other trees. We now show that $\lceil \frac{h}{\log_d B} \rceil$ is indeed an upper bound for all trees.

Theorem 2. *Let $T$ be an arbitrary $N$-node tree of height $h$ in which a node has at most $d$ children. Then, $bl^* \leq \lceil \frac{h}{\log_d B} \rceil$.*

**Proof:** We again assign subtrees to blocks, with the root of $T$ being the root of the first subtree. Assume node $v$ is the root of a subtree to be assigned to a block. Let $T_v$ be the subtree of $T$ rooted at $v$. We include at least the next $\log_d B$ levels of subtree $T_v$. If $T_v$ contains fewer than $\log_d B$ levels, the block contains all nodes in subtree $T_v$ and it is not complete. Once $\log_d B$ levels of $T_v$ have been included, we continue to include nodes of $T_v$ level-by-level until the block is complete. Hence, compared to complete trees, the blocknumber can only decrease and the bound follows. □

The mappings described above make no effort to assign $B$ nodes to a block and are thus not complete. We next describe how to transform these mappings into complete ones. The transformations increase the blocknumber by at most one.

Lemma 3. *Let $P$ be a mapping of tree $T$ to blocks of size $B$ having blocknumber $bl$. Then, there exists a complete mapping $P'$ having blocknumber at most $bl + 1$.*

**Proof:** The complete mapping $P'$ is obtained from $P$ in two phases. First, the blocks of mapping $P$ are considered level-by-level starting with the block containing the root of $T$. If a block, say block $X$, contains fewer than $B$ nodes and not all leaves in $X$ correspond to leaves of $T$, children of leaves in $X$ are assigned to $X$. 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 terminates once all blocks have been processed.

Assume the reassignment phase generates $m$ blocks containing fewer than $B$ nodes. Let $B_1, B_2, \ldots, B_m$ be these blocks and $n_1, n_2, \ldots, n_m$ be the number of nodes in these blocks, with $n_1 \leq n_2 \leq \ldots \leq n_m < B$. The second phase generates a complete mappring by reassigning subtrees. The reassignments may increase the blocknumber by one. 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 $n_i + \ldots + n_{i+j-1} \leq B$ and $n_i + \ldots + n_{i+j} > B$. 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 both phases have been applied to the initial mapping $P$, at most one block contains fewer than $B$ nodes and we thus have a complete mapping $P'$. The blocknumber increased by at most one. This happens on a path from the root to a leaf which was reassigned during phase 2. □

We can thus conclude with the following corollary.

COROLLARY 4. *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

We now turn to the efficient generation of complete blocks when nodes of $T$ can be replicated and an upper bound on the total number of nodes is given. We describe an algorithm which generates a complete mapping for $\tau \leq \frac{3}{2}$ having a blocknumber of at most $\lceil \frac{h - \log_d B}{B} \rceil + 2$. This compares favorably to the bound of $O(h / \log_d B)$ which is the best possible for some trees when $\tau = 1$. The mapping can be generated in $O(N)$ time. Related work on replication is discussed in [7]. In particular, the paper describes a mapping of a complete binary tree of height $h$ for $\tau = 2$ achieving a blocknumber of $\lceil \frac{h}{\log B} \rceil$.

Our algorithm consists of three phases: preprocessing, path creating, and balancing. We first discuss each one of the three phases in detail. We then discuss an $O(N)$ time main memory implementation.

**PHASE 1: Preprocessing.**
For every node $v$ in $T$ we first determine the quantity $desc(v)$, 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 $desc(v) \leq 2B$ and $desc(p(v)) > 2B$. A subtree leaf represents a subtree of $T$ containing at most $2B$ nodes. Such a node $v$ and its descendents form one block. Finally, the descendents of a subtree leaf $v$ are removed from $T$ and the resulting new tree is used in Phase 2.

Figure 1(a) shows a tree $T$ with $B = 6$ after the preprocessing phase. Subtree leaves are represented as cubes. The integer on the cube represents the number of nodes in the subtree (and stored in the corresponding block). For $B = 6$, 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 the preprocessing phase. The path-creating phase consists of a bottom-up traversal of $T'$ in which subtrees of height $B$ are identified and removed from $T'$. The last step of the traversal may generate a subtree of height less than $B$. The subtrees are used to generate paths which are assigned to blocks.

Let $H$ be a generated subtree of $T'$ having height $B$. In the first subtrees identified by the path-creating phase, 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). Let $r_H$ be the root of subtree $H$. 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, subtree $H$ - with the exception of root $r_H$ - is removed from $T'$. Node $r_H$ is now a leaf and is marked as a *path leaf.*

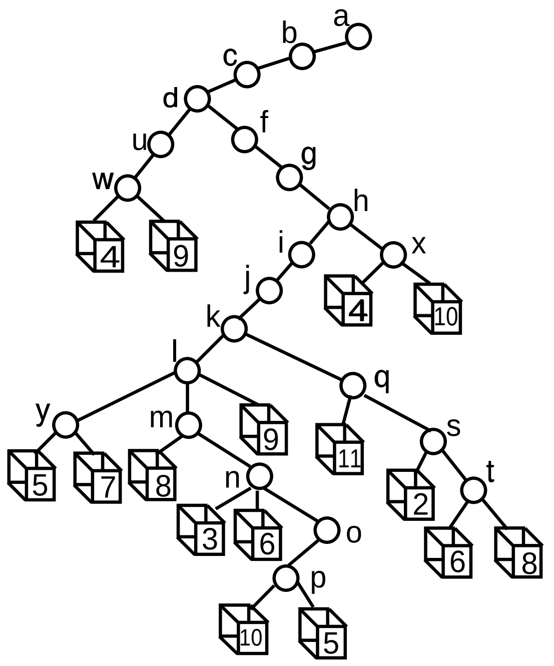
When applying the path-creating phase to the tree shown in Figure 1(a), we first identify node $k$ as the root of a subtree $H$ of height $B = 6$. The path of length 6 from $k$ to $p$ forms one complete block, block $B_p$. Subtree $H$ contains two other end-nodes, nodes $y$ and $t$, and we generate two more blocks $B_y$ and $B_t$, as shown in Figure 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. This process results in making three copies of node $k$. The subsequent steps of the path-creating phase view node $k$ as a path leaf. Four more blocks, namely blocks $B_j$, $B_x$, $B_w$, and $B_c$, are generated.

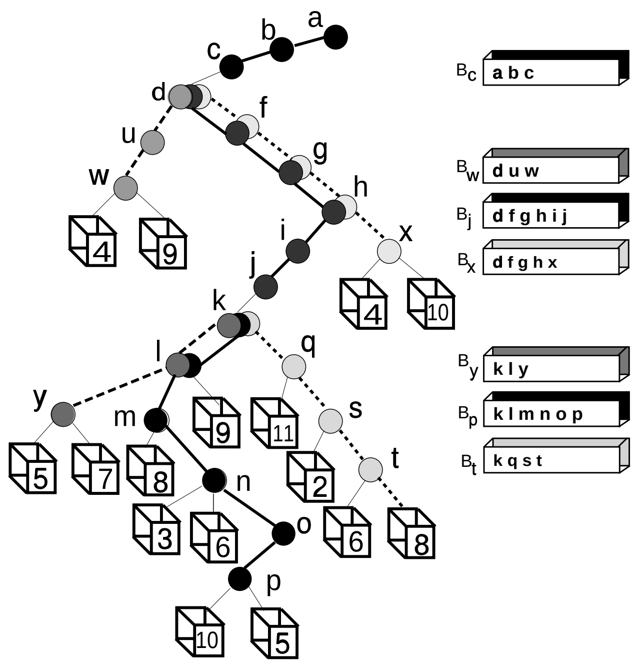
**PHASE 3: Balancing.**
After the path-creating phase, every block generated so far contains between 1 and $2B$ nodes of T and it represents either a path or a subtree. A block assigned a path contains at most $B$ nodes and a block assigned a subtree contains at most $2B$ nodes. The balancing phase generates a complete mapping by reassigning nodes. The reassignments will 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 considered by this top-down traversal 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.

• *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.

Figure 2 illustrates how the balancing phase changes blocks formed by the path-creating phase. As indicated in Figure 1(b), block $B_c$ initially contains three nodes, including root $a$ and $c$ as the end-node. Block $B_j$ contains 6 nodes


(a) tree $T$ after preprocessing; cubes represent subtree leaves.



(b) after path-creating; blocks created are indicated in tree and are shown explicitly.

**Figure 1: Illustrating phases 1 and 2 for $B = 6$.**

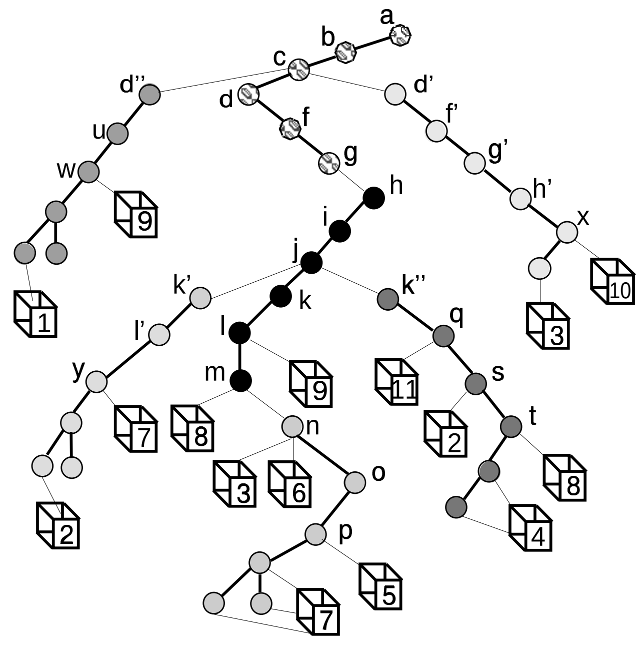


**Figure 2: Illustrating phase 3 for $B = 6$.**

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 $B_p$. Block $B_p$ becomes complete by taking three nodes from subtrees rooted at the children of $p$. The process of assigning nodes from subtrees to incomplete blocks is described next.

• *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.

Continuing the discussion of Figure 2, the process described above identifies the three nodes assigned to block $B_p$. End-node $p$ has two children, one representing a subtree of size 10 and one of size 5. In the solution shown in Figure 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 Figure 3. The light edges in Figure 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. To complete the balancing phase, blocks containing subtrees are made complete. These blocks contain between 1 and $2B$ nodes. We first consider the blocks containing more than $B$ nodes. For

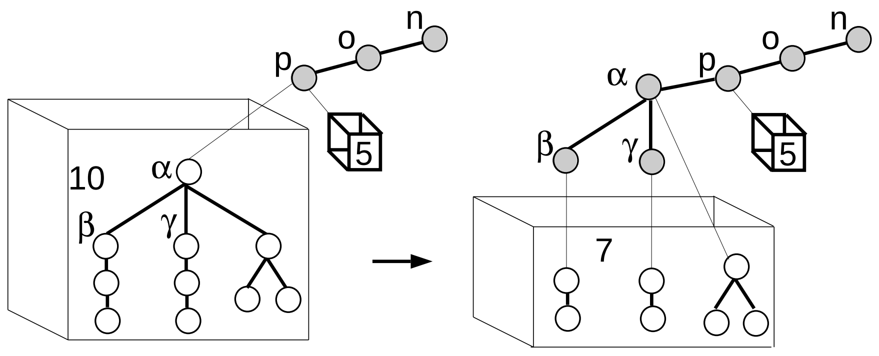


**Figure 3: Making block $B_p$ complete: nodes $\alpha$, $\beta$, and $\gamma$ are assigned to $B_p$.**

each such block, we identify 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 strategy underlying Lemma 3. Reassignments made increase the blocknumber by at most 1.

We next sketch an $O(N)$ time internal memory algorithm. Assume tree $T$ is stored so that for every node $v$, $v$'s parent and a list of $v$'s children are available. The preprocessing phase corresponds to traversing $T$ in $O(N)$ time, starting at the root, and identifying all subtree leaves. The path-creation phase corresponds to a bottom-up traversal of the tree which starts at the subtree leaves. The traversal performs the necessary height computations to identify the roots of subtrees used for generating the paths assigned to blocks. For every root $r_H$ of such a subtree $H$, we generate all paths from $r_H$ to end-nodes in $H$ and then mark $r_H$ as a path leaf. In addition to generating the paths, the path-creation phase generates for every path leaf $r_H$

- links to the blocks containing node $r_H$ and
- a link to one block containing a path of length exactly $B$.

At least one such block exists for every $r_H$, except for the root of $T$. For every subtree leaf $v$, the traversal generates a link to the one block containing the subtree rooted at $v$. The total amount of work done in the path-creation phase is $O(N)$.

The balancing phase traverses tree $T$ again in a top-down fashion. For every node of $T$ identified as a root $r_H$ of a subtree, the traversal generates complete blocks containing $r_H$ as follows. For every block $Bl$ containing node $r_H$, determine the other end-node on the path stored in $Bl$. Let this be node $e$ and let $r_{H'}$ be a child of $e$ in $T$. If $r_{H'}$ is a path leaf, node $r_{H'}$ of $T$ has access to a block $Bl'$ containing $B$ nodes. We use block $Bl'$ to make block $Bl$ (which contains nodes $r_H$ and $e$) complete, as described by the algorithm. Block $Bl'$ now contains a shorter path and one end-node changed. Observe that for generating a complete mapping we do not need to update information about the new end-node of this path. The described process of making a block complete can be applied block $Bl$ complete. When all children of node $e$ are subtree nodes, the balancing follows the description given earlier in a straightforward way. Overall, balancing and thus the generation of the complete mapping takes $O(N)$.

Theorem 5. *The nodes of an arbitrary $N$-node tree $T$ can be mapped to complete blocks using at total replication factor $\tau < 3/2$ and achieving a blocknumber of $O(\frac{h}{B})$. The mapping can be generated in $O(N)$ time by an internal memory algorithm.*

**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 $\tau < 3/2$. 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 $|P| \leq B$ 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.

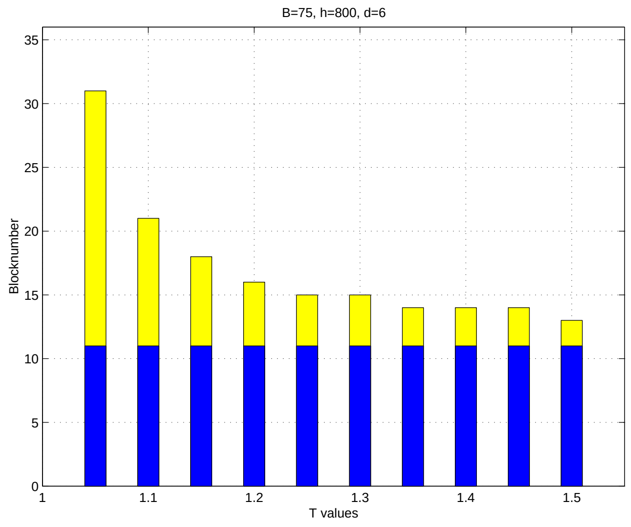


**Figure 4: Maximum blocknumber in terms of contributions by the path-creating (dark) and balancing (light) phase as $\tau$ changes.**

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 $\frac{3}{2}N$. $\square$

Since a total replication factor of 3/2 results in an asymptotically optimal blocknumber, we do not consider larger replications. The obtained result can be generalized to smaller replication factors and a tradeoff between total replication and blocknumber can be established. For arbitrary $\tau$, $1 \leq \tau < 3/2$, the preprocessing phase identifies the subtree leaves as the nodes $v$ which have $desc(v) \leq \frac{1}{\tau-1}B$ and $desc(p(v)) > \frac{1}{\tau-1}B$. The path-creating phase generates a mapping in which a block contains between 1 and $\frac{B}{\tau-1}$ 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 $\lceil \frac{1}{\tau-1} \rceil$. Figure 4 shows the increase in the blocknumber as $\tau$ approaches 1. Observe that this plot gives an upper bound on the blocknumber and that this upper bound is the best possible for complete trees. In summary, using $O(\tau N)$ space, we can generate a complete mapping of an arbitrary tree achieving a blocknumber of at most $\lceil (h - \log_d \frac{1}{\tau-1}B)/B \rceil + \lceil \frac{1}{\tau-1} \rceil$.

We conclude this section by pointing out a limitation of the mapping. When a search in tree $T$ proceeds towards the root (e.g., from a leaf to the root), information can be added to each block which identifies the next block needed. In this case, the blocknumber of the tree corresponds to the blocknumber used by the actual search. For other types of tree searches, the mapping described has some limitations. Additional information would be needed to identify blocks and to make the blocknumber of of mapping correspond to the blocknumber of the search. We point out that this is not the focus of our work. We concentrated, in this and the next section, on showing that there exist mappings achieving a good blocknumber and on how replication impacts the blocknumber.

## 4. MAPPINGS WITH A NODE REPLICATION BOUND

The previous section described an algorithm having total replication factor $\tau \leq 3/2$. In this mapping, the number of times a node is replicated depends on the structure of $T$ and on $B$. In this section we consider complete mappings in which every node is replicated at most $\rho$ times, $\rho \geq 2$. While for a constant total replication factor we can show that there exist mappings achieving an asymptotically optimal blocknumber, a constant node replication factor brings no significant reduction in the blocknumber. In particular, we can show that any mapping of an $N$-node complete binary tree with node replication $\rho$ has a blocknumber of at least $\lceil \frac{h}{\log \rho + \log B} \rceil$.

For complete binary trees, a blocknumber of $\frac{h}{\log \rho + \log(B - \log \rho)}$ can be achieved by using the following approach which also underlies our solution for general trees. Assign to every block a path $P$ and a subtree $S$ rooted at the end-node of path $P$. Nodes on the path experience replication, with the start-node of path $P$ having the highest node replication (i.e., this node appears in up to $\rho$ blocks). Nodes in $S$ have a replication of 1. Choose $P$ and $S$ such that $|P| + |S| \leq B$ and the start-node of $P$ is replicated at most $\rho$ times. Paths are created similar to the path-creating done for total node replication. For complete binary trees we choose $|P| = \log \rho$ and let $S$ to be a complete binary tree containing $B - \log \rho$ nodes. This approach makes at most $\rho$ copies of every node and results in a blocknumber of $\frac{h}{\log \rho + \log(B - \log \rho)}$. Using an argument similar to the one used in Theorem 2, we can show that this upper bounds holds also for arbitrary binary trees.

The remainder of this section describes an algorithm generating a complete mapping for arbitrary trees having node replication at most $\rho$. The algorithm has an $O(\rho N)$ internal memory time bound. While for many binary trees this algorithm achieves a blocknumber better than the upper bound stated in the above paragraph, we do not know of a general better upper bound.

In the mapping generated, every block contains a path and a subtree rooted at the end-node of the path. Nodes on the path are replicated nodes of $T$, with the start node of the path having the highest replication. Assume node $u$ of tree $T$ is replicated $rep(u)$ times and $u$ has $k$ children. The algorithm uses information on the current replication, blocknumber, and block utilization to decide how many replications can be used for each child of $u$. Each replication corresponds to one path containing node $u$. For all paths containing children of $u$, the algorithm decides which paths to continue (by adding node $u$) and which to terminate. The complete mapping is generated in two phases. In the block-creating phase we traverse tree $T$ in a bottom-up fashion. We bring up information about subtrees which allows us to determine which paths to continue and where to terminate a path. The termination of a path creates a block containing a path with at most $B$ nodes on it. The balancing phase generates a complete mapping by assigning to blocks a path as well as a subtree rooted at the end-node of the path.

**PHASE 1: Block-Creating.**
The block-creating phase traverses tree $T$ from the leaves towards the root. Assume node $u$ has children $u_1, \dots, u_k$. Assume we have determined the following entries for every child $u_i$ of $u$:

- $bl(u_i)$, the maximum blocknumber of the mapping generated for the subtree rooted at node $u_i$,
- $rep(u_i)$, the replication number of node $u_i$ in this mapping, $1 \leq u_i \leq \rho$,
- $m\_len(u_i)$, the maximum length of a path containing node $u_i$ (the length of a path is the number of nodes on it), $1 \leq m\_len(u_i) \leq B$.

Let $m\_bl = \max_{1 \leq i \leq k} bl(u_i)$. For all children $u_i$ with $bl(u_i) < m\_bl$, the continuation of the started paths would use up node replications for node $u$, but it would not help decreasing the current blocknumber. To avoid this "waste" of node replications, we terminate every path at $u_i$ when $bl(u_i) < m\_bl$; i.e., we create $rep(u_i)$ blocks containing a path with node $u_i$ is the start node. Assume that children of $u$ with $bl(u_i) < m\_bl$ have been handled. Rename the remaining children of $u$ so that the children are $u_1, \dots, u_l$ and for every child we have $bl(u_i) = m\_bl$, $1 \leq i \leq l$. How many copies to make of $u$ and which paths to assign the copies is decided as follows:

- Case 1: All paths having maximum blocknumber have length less than $B$ and can be continued.
  If for every child $u_i$ we have $m\_len(u_i) < B$ and $\sum_{1 \leq i \leq l} rep(u_i) \leq \rho$, we set $rep(u) = \sum_{1 \leq i \leq l} rep(u_i)$, $bl(u) = m\_bl$, and $m\_len(u) = \max_{1 \leq i \leq l} m\_len(u_i) + 1$. This case applies to node $u$ shown in Figure 5 for which a total of five paths are continued. Observe that the three paths containing node $u_4$ are terminated since the blocknumber for the subtree rooted at $u_4$ is only 3.
- Case 2: Not all paths of maximum blocknumber can be continued.
  If $\sum_{1 \leq i \leq l} rep(u_i) > \rho$, at least one path having a child of $u$ as a start-node has to be terminated. Terminating a path results in a blocknumber of $m\_bl+1$ for vertex $u$. Hence, there is no advantage in continuing any of the paths. We terminate all paths having the children of $u$ as a start-node and set $rep(u) = 1$, $bl(u) = m\_bl + 1$, and $m\_len(u) = 1$.
- Case 3: One path contains $B$ nodes.
  If there exists a child $u_i$ such that $m\_len(u_i) = B$, node $u_i$ is in a block which is complete. Nodes $u$ and $u_i$ are thus placed into different blocks and this increases the blocknumber by 1. As for case 2, there is no advantage in continuing any of the other paths. We set $rep(u) = 1$, $bl(u) = m\_bl + 1$, and $m\_len(u) = 1$.

Let $P$ be the mapping generated by the path-creating phase. Every block of mapping $P$ contains a path with at most $B$ nodes on it and every node is replicated at most $\rho$ times. Let $bl$ be the blocknumber of $P$. The subsequent balancing phase generates a complete mapping with a blocknumber of at most $bl + 1$.

**PHASE 2: Balancing.**
The balancing phase starts by considering the blocks containing a copy of the root $r$ of $T$. Initialize set $A$ to contain all block having a copy of $r$. Choose and delete an arbitrary block, say block $B_i$, from set $A$. Let $f_i$ be the start-node on the path in block $B_i$ and $l_i$ be the end-node on this path. To make block $B_i$ complete, we use tree $T$ to determine the $B - |B_i|$ nodes added to block $B_i$: starting with node $l_i$, traverse, in $T$, the subtree rooted at $l_i$ in a level-by-level fashion and include $B - |B_i|$ nodes. Next, consider all the blocks containing at least one of the $B - |B_i|$ selected nodes. For a block $B_u$ containing such a node $u$, we delete from block $B_u$ node $u$ as well as all the nodes from $u$ to the start-node of the path in $B_u$. Block $B_u$ is then added to set $A$.

The process of adding subtrees to blocks and shortening paths continues until blocks containing a leaf node are encountered. At this point, every block which does not contain a leaf of $T$ is a complete block. Since nodes move to blocks closer to the root, the reassignments made so far do not increase the initial blocknumber. Let $B_1, B_2, \dots, B_r$ be the remaining non-complete blocks. Assume we are processing block $B_i$ and have processed $B_1, \dots, B_{i-1}$, $1 \leq i \leq r$. If the current number of nodes in $B_i$ is equal to $B$, $B_i$ is not changed and we continue with $B_{i+1}$. Assume thus that $|B_i| < B$. We assign nodes from $B_{i+1}, B_{i+2}, \dots$ to block $B_i$ until $B_i$ contains exactly $B$ nodes. If the last block giving nodes to $B_i$ is $B_{i+j}$, then blocks $B_{i+1}, \dots, B_{i+j-1}$ no longer exist and $B_{i+j}$ has a new start-node. The process continues with block $B_{i+j}$. Once completed, all blocks, with the exception of block $B_r$, are complete. The blocknumber may have increased by 1 and thus be $bl + 1$. This concludes the description of the algorithm for generating a complete mapping with a given maximum node replication.

Unlike to the total node replication case, we are not able to show a non-trivial bound on the blocknumber achieved. For arbitrary binary trees, the blocknumber generated is bounded by $O(\frac{h}{\log \rho + \log(B - \log \rho)})$. For arbitrary binary trees and no node replication, Theorem 2 gives $O(\frac{h}{\log B})$ as an upper bound on the blocknumber. Thus, for constant $\rho$, there is no provable asymptotic improvement compared to allowing no node replication. In contrast, constant total node replication allows an asymptotically optimal blocknumber.

## 5. 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 was measured in terms of the blocknumber achieved and the amount of repli-

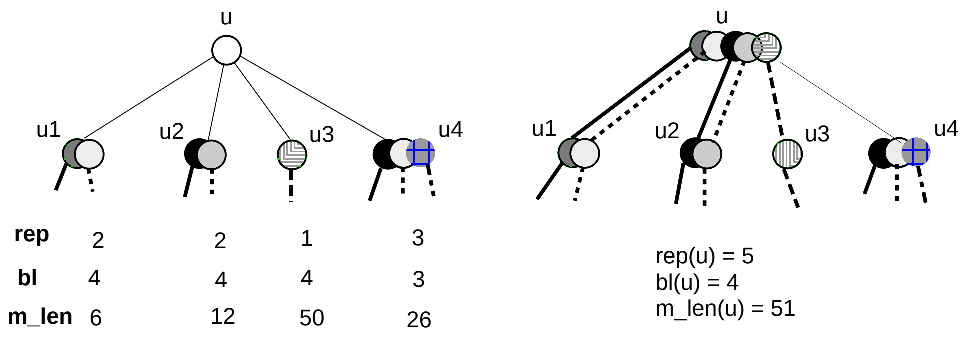


**Figure 5: Node $u$ before and after applying Case 1 when $\rho = 6$; $u$ receives a replication of 5.**

cation. We considered total node replication and individual node 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 $\lceil h/B \rceil$. In this complete mapping, individual node replication depends on the structure of the tree and can be large. When the individual node replication is bounded, the blocknumber cannot be reduced as much. This is an indication that for search and access structures with node replication, bounding the total space is more effective than bounding individual node replication. 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.

## 6. REFERENCES

[1] Y. Breitbart and H. Korth. Replication and consistency: being lazy helps sometimes. In *Proc. of 16-th ACM Symp. on Principles of Database Systems*, pages 173–184, 1997.

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

[3] A. W.-C. Fu and D. W.-L. Cheung. A transaction replication scheme for a replicated database with node autonomy. In *Proc. of 20th International Conference on Very Large Data Bases*, pages 214–225, 1994.

[4] J. Gil and A. Itai. How to pack trees. *Journal of Algorithms*, 32, 1999.

[5] J. Hellerstein, E. Koutsoupias, and C. Papadimitriou. On the analysis of indexing schemes. In *Proc. of 16th ACM Symp. on Principles of Database Systems*, pages 249–256, 1997.

[6] M. Herlihy. Concurrency and availability as dual properties of replicated atomic data. *Journal of the ACM, JACM*, 37(2):257–278, Apr. 1990.

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

[8] S. Ramaswamy and S. Subramanian. Path caching: A technique for optimal external searching. In *Proc. of 13th ACM Symp. on Principles of Database Systems*, volume 13, pages 25–35, 1994.

[9] H. Samet. *Applications of Spatial Data Structures, Computer Graphics, and Image Processing.* Addison-Wesley, 1990.

[10] S. Subramanian and S. Ramaswamy. The P-range tree: A new data structure for range searching in secondary memory. In *Proc. of the 6th Annual Symposium on Discrete Algorithms*, pages 378–387, Jan. 1995.

[11] P. Triantafillou and F. Xiao. Supporting partial data accesses to replicated data. In *Proc. of the 10th International Conference on Data Engineering*, pages 32–42, Feb. 1994.

[12] J. S. Vitter. External memory algorithms. In *Proc. of 17th ACM Symp. on Principles of Database Systems*, pages 119–128, 1998.

[13] O. Wolfson, S. Jajodia, and Y. Huang. An adaptive data replication algorithm. *ACM Transactions on Database Systems*, 22(2):255–314, June 1997.

[14] O. Wolfson and A. Milo. The multicast policy and its relationship to replicated date placement. *ACM Transactions on Database Systems*, 16(1):181–205, Mar. 1991.