ROBUST BALANCING IN B-TREES

Extended Abstract

Scott Huddleston
Information and Computer Science
University of California at Irvine
Irvine, California 92717
USA

Kurt Mehlhorn
Fachbereich 10 - Informatik
Universität des Saarlandes
6600 Saarbrücken
Federal Republic of Germany

Abstract

Robust balancing is a technique for maintaining generalized B-trees with a cumulative rebalancing cost that is asymptotically linear. It is especially significant in conjunction with fingers, which can make cumulative search cost linear. We define a new family of robust balancing algorithms which includes algorithms with sublinear rebalancing cost, extending previous work that has shown linear algorithms. An accounting technique is developed which gives strong bounds on rebalancing cost. We also consider robust balancing for B^*-trees, a B-tree variant with improved space utilization.

1. Introduction

Robust balancing is a technique for maintaining generalized B-trees with a cumulative rebalancing cost that is asymptotically linear. This contrasts with "fragile balancing" in the standard B-tree algorithm: a worst case rebalancing cost of $\Theta(k \log n)$ for k intermixed insertions and deletions to a tree of size n.

Robust balancing is especially significant in conjunction with certain finger schemes [BT80,HM80], which give fast and flexible searching (linear total search cost in many applications), but do not affect balancing cost. Robust balancing with these schemes gives an order of magnitude improvement in total update cost.

Previous work [Hu79,MS79,HM80] has shown linear bounds on rebalancing cost with robust balancing. We will give new algorithms with sublinear bounds on rebalancing cost (fewer than k/c rebalancing operations for k updates, where $c > 2$ and can be quite large). Sublinear bounds are significant for B-trees stored on disk, even though there is a linear lower bound on leaf accesses, because rebalancing operations require several more disk accesses than leaf updates, and can be very expensive for B-trees augmented with additional links [HM80]. An optimal sublinear algorithm can improve total update cost over a linear algorithm by a substantial constant factor.

We also consider robust balancing algorithms for B^*-trees, a generalization of B-trees with improved space utilization. These algorithms give new tradeoffs among search cost, rebalancing cost, and space utilization, and provide further reasons to favor robust balancing over fragile balancing.

Section 2 describes a new family of robust balancing algorithms. Section 3 presents a powerful accounting technique for analyzing rebalancing cost in B-trees. Section 4 derives strong upper bounds and some tight bounds on asymptotic rebalancing cost. Section 5 analyzes rebalancing cost for arbitrary initial trees, and discriminates asymptotically equivalent algorithms. Section 6 considers B^*-trees. Most of these results are treated in depth in [Hu81].

2. Robust B-Trees and Algorithms

The generalization of B-trees that makes robust balancing possible is to extend the allowable range of arities (i.e. number of sons) of nodes. Let $|T|$ be the number of leaves of tree T, and let an interior node be an internal node other than the root. We define a (u,v)-tree, for $2 \le u \le \lceil v/2 \rceil$, to be a multiway tree in which all root-to-leaf paths have the same length, every interior node has between u and v sons inclusive, and the root has between $\min(2,|T|)$ and v sons. We define $r = v+1-2u$ to be the robustness of (u,v)-trees, and (following [MS79]) $p = \lceil v/2 \rceil - u$ to be the hysteresis. A robust B-tree is a (u,v)-tree with $v \ge 2u$. It is immediate that the height h of a (u,v)-tree T is $\Theta(\log|T|)$, since for $h>1$, T has at least $2u^{h-1}$ and at most v^h leaves.

We assume familiarity with the standard (fragile) B-tree algorithm and rebalancing operations. The (u,v)-tree algorithms we consider use the same control structures and types of rebalancing operations as the standard algorithm. However, there is much more flexibility in choosing which rebalancing operations to apply-robust balancing exploits this fact.

Algorithms for (u,v)-trees are determined by four parameters. The merge threshold t, $0 \le t \le v-2u+1$, specifies whether to merge or share on underflow: merge if the adjacent brother used with underflow has arity $\le u+t$, and share otherwise. The underflow context selects which adjacent brother to use when underflow occurs. One-sided context can examine and use only one adjacent node in underflow, and two-sided context can examine two adjacent brothers before selecting one. The splitting strategy we use partitions arities of the resulting nodes as evenly as possible. We consider two sharing strategies: strategy Z_s for $s \ge 1$ always shifts s sons to an underflow node (or for "skewed" shifting, reverses the order of the resulting node arities); strategy Z_0 shifts to partition resulting node arities nearly evenly. Other values of these parameters, such as uneven splitting or a nondeterministic range of t, will not be considered here.

Let $Alg(t)$ denote the class of all algorithms for a given class of (u,v)-trees with merge threshold t. $Alg(t,Z_i)$ for $i \ge 0$ is the class which also uses sharing strategy Z_i. $Alg(t,s_{min})$ is the class of algorithms with merge threshold t and shift hysteresis s_{min}. The value s_{min} is defined for any sharing strategy in

(u,v)-trees to be the minimum, over all share operations, of min(x,y)-u+1, where x and y are the two node arities created by sharing. Thus $s_{min} = \min(s,t+2-s)$ for strategy Z_s, and $1+\lfloor t/2 \rfloor$ for strategy Z_0.

3. An Accounting Technique

We use an accounting technique to analyze rebalancing cost which follows the "bank account" paradigm. This paradigm associates operations on a data structure with deposits and withdrawals in an account. It proceeds by constructing accounts V(S) (functions from sets of nodes to integers) such that updates (insertions or deletions) increase V(S) by at most 1, and rebalancing operations decrease V(S) by at least w units for some w. The withdrawals can also be adjusted to account for different weights on rebalancing operations, such as disk access cost.

An account V(T) for tree T is simply the sum of V(x) over internal nodes x of T, where V(x) depends only on the arity of x and whether or not x is the root. We define an account V for a (u,v)-tree algorithm by giving two functions, $V(j) = V^I(j)$ on interior node arities $u-1 \le j \le v+1$, and $V^R(j)$ on root node arity $0 \le j \le v+1$. An account V with $V(S) \ge 0$ for all sets S we call a savings account.

Define B(k,T,S) to be the total number of rebalancing operations that occur during the first k updates S_k of an update sequence S on initial tree T. We often write B(k) for B(k,T,S). Most of our upper bounds on B(k) will be shown by a standard technique, the savings account argument. Let V be a savings account for some (u,v)-tree algorithm. Then we have $0 \le V(T_k) \le V(T_0) + k - wB(k)$ on k updates to initial and final trees T_0 and T_k. This gives $B(k) \le 1/w\ (k + V(T_0))$ $= k/w + O(|T_0|)$, and $B(k) \le k/w$ when T_0 is the empty tree.

4. Asymptotic Rebalancing Cost

We first derive strong upper bounds on asymptotic rebalancing cost using the savings account argument. These bounds are shown by elegant savings accounts, often of the form $V = V_{z,b}$, defined on (u,v)-trees by

$$V^I(j) = b + |j-z| \quad \text{for } u-1 \le j \le v+1; \qquad V^R(j) = \max(0, b+j-z) \quad \text{for } 0 \le j \le v+1.$$

Theorem 1 gives our best upper bound on rebalancing cost, which is realized when parameters t and s_{min} are sufficiently large. Theorem 2 shows that sharing in preference to merging whenever underflow occurs gives a linear algorithm for all robust B-trees. Theorem 3 considers rebalancing cost for nonoptimal values of t and s_{min}. Theorem 4 treats the case of weighted rebalancing operations. Theorem 5 examines the distribution of rebalancing cost over the levels of a tree.

We also indicate how savings accounts can help in finding good lower bounds, and use this approach to show that the bounds of Theorems 1 and 2 are tight for odd-order (u,v)-trees and (2,4)-trees.

Theorem 1: $B(k) \geq k/p + O(|T|)$ for any (u,v)-tree algorithm in $Alg(t,s_{min})$ on initial tree T with $v \geq 2u+1$, $t \geq \lceil 3p/2 \rceil - 1$, and $s_{min} \geq \lceil p/2 \rceil$, where $p = \lceil v/2 \rceil - u \geq 1$ is the hysteresis of (u,v)-trees.

Proof: Let savings account V be $V_{m,p+1}$, where $m = u+p$. We will show that each rebalancing operation decreases V by at least p (i.e. if T' and T" are the partially rebalanced trees before and after the rebalancing operation, then $V(T')-V(T") \geq p$). With this fact established, the theorem follows by the savings account argument.

Let $p' = p+1$, $v' = v+1$, and $u' = u-1$. For splitting, suppose a v'-node x is split into nodes of arities $i = \lfloor v'/2 \rfloor = m$ and $j = \lceil v'/2 \rceil$. Since V of x's father can increase by at most 1,

$$V(T')-V(T") \geq V(v')-V(i)-V(j)-1 = (v'-p')-(i-p')-(j-p')-1 = p'-1 = p.$$

For merging, suppose a u'-node x is combined with an (m+i)-node y, $-p \leq i \leq t-p$, to form a j-node z, $j = u'+m+i$. Note that x must be an interior node, and $V(x)=m=u'+p'$. Also note that z might be the root, in which case $V(z)=V^R(j) \leq V^I(j)=V(j)$. Then

$$V(T')-V(T") \geq V(u')+V(m+i)-V(j)-1 = (u'+p')+(m+|i|-p')-(m+|u'+i|-p')-1$$
$$= |u'|+|i|-|u'+i| + p'-1 \geq p'-1 = p.$$

For sharing, suppose an i-node shifts sons to a u'-node, creating a j_1-node and a j_2-node. Then $u'+s_{min} \leq j_1, j_2 \leq i-s_{min}$ by the definition of s_{min}. Also $V(j_1)+V(j_2) \leq V(u'+s_{min})+V(i-s_{min})$, since $j_1+j_2 = u'+i$ and V is a concave function. Thus

$$V(T')-V(T") = V(u')+V(i)-V(j_1)-V(j_2)$$
$$\geq (V(u')-V(u'+s_{min})) + (V(i)-V(i-s_{min})) = 2s_{min} \geq 2\lceil p/2 \rceil \geq p.$$

This completes the proof of Theorem 1.

Theorem 2: $B(k) \geq 3k/2 + O(|T|)$ for any (u,v)-tree algorithm, $v \geq 2u$, in Alg(0).

Proof sketch: By the savings account argument, using account V defined by

$$V^R(j) = \max(0, j-(2u-1)); \quad V^I(j) = \max((u+1/3)-j, V^R(j)).$$

Then $V^I(j) = [4/3, 1/3, 0, \ldots, 0, 1, 2]$ when $j = [u-1, u, u+1, \ldots, v-1, v, v+1]$, and each rebalancing operation decreases V by at least 2/3.

Theorem 3: Consider a (u,v)-tree algorithm in $Alg(t,s_{min})$ with $v \geq 2u+1$, $p = \lceil v/2 \rceil - u$, and $t < \lceil 3p/2 \rceil - 1$ or $s_{min} < \lceil p/2 \rceil$. Let $w = \min(2(t+1)/3, 2s_{min})$. Then $B(k) \leq k/w + O(|T|)$.

Theorem 3 is shown by the savings account argument using savings account $V = V_{z,b}$, where $b = 1+w$ and $z = u + \min(p, 2(t+1))$. We remark that Theorem 3 also applies to the special case of v even and t=p, using $z = u+2(t+1)/3$, where $z > u+p$ for $p \leq 1$. This case independently establishes Theorem 2, and improves the bound on B(k) for Alg(1) on (u-1,2u)-trees.

For weighted rebalancing operations, define B(k) to be the sum of rebalancing operation costs. Theorem 4 shows better bounds for weighted rebalancing cost than would be obtained by simply using the preceding results and dividing by the maximum

rebalancing weight.

Theorem 4: Let split, merge, and share operations have costs c_1, c_2, and c_3 resp. Consider a class of (u,v)-trees with hysteresis $p < \lceil v/4 \rceil$, and a (u,v)-tree algorithm in $Alg(t,s_{min})$. Let $c' = (c_1+c_2)/2$, $d = 1+c_3/2c'$, and $w = \min((t+1)/d, 2c's_{min}/c_3)$. Then

a) $B(k) \leq c'k/p + O(|T|)$ if $t \geq \lceil dp \rceil - 1$ and $s_{min} \geq \lceil c_3 p/2c' \rceil$,

b) $B(k) \leq c'k/w + O(|T|)$ if $t < \lceil dp \rceil - 1$, or $t=p$ and v is even.

Proof sketch: We use a savings account $V_{z,b}$, similar to Theorem 3, and adjust parameters z and b to account for relative rebalancing weights. The condition $p < \lceil v/4 \rceil$ ensures that V is nonnegative, and can be dropped if $c_1 \leq c_2$.

Theorem 5 shows that the distribution of rebalancing operations over the levels of a tree decreases exponentially with height. This is important for reducing locking contention in multiprocessing environments [BS77], and for trees used in multidimensional applications [Wi77], where the cost of rebalancing operations increases exponentially with height. We say that a rebalancing operation caused by a level h underflow or overflow node "occurs at level h", where leaves have level 0.

Theorem 5: Let $B^h(k)$ be the number of rebalancing operations that occur at level h during a sequence of k updates to an initially empty tree. For the class of (u,v)-trees with robustness $r = v-2u+1$ and hysteresis $p = \lfloor r/2 \rfloor$, consider an algorithm in $Alg(t,s_{min})$. Then

a) $B^h(k) \leq k/(p+1)^h$ if $t \geq \lfloor 3p/2 \rfloor$ and $s_{min} \geq 1+\lfloor p/2 \rfloor$,

b) $B^h(k) \leq k(3/5)^{h-1}$ in (2,4)-trees with $t = 0$,

c) $B^h(k) \leq k/(r+1)^{h-1}$ if $t = 0$ and $u \geq \lceil (v+3)/4 \rceil$.

Proof sketch: We use the savings account argument, restricted to nodes at one level. Let $W^h(k)$ be the number of level h node expansions and contractions during k updates, and $SM^h(k)$ be the number of level h splittings and mergings. Then

(1) $W^1(k) = k$; $W^{h+1}(k) = SM^h(k)$ for $h \geq 1$.

For cases (a) and (b), let V be the savings account defined in Theorems 1 and 2 resp. For case (c), let $p' = \lceil r/2 \rceil$, and define a savings account V by

$V^R(j) = \max(0, j-(v-2p'))$; $V^I(j) = \max((u+p')-j, V^R(j))$.

Let $w = p+1$ for case (a), 5/3 for case (b), and r+1 for case (c). Then the savings account argument applied to level h splittings and mergings, with the fact that sharing does not increase V^h, shows that $SM^h(k) \leq W^h(k)/w$. Together with (1) and induction on h, this shows

(2) $W^h(k) \leq k/w^{h-1}$.

Now $B^h(k) \leq W^h(k)$, since expansion or contraction must precede rebalancing. Thus (2) establishes parts (b) and (c) of the theorem. Part (a) is shown by the savings account argument applied to all level h rebalancing operations, which follows the proof of Theorem 1 closely.

Savings accounts also provide a useful approach for demonstrating good lower bounds: try to construct a sequence of updates in which every leaf operation increases V by 1, and every rebalancing operation decreases V by exactly the minimum amount. We show two cases in which the preceding upper bounds are tight. In other cases one can show lower bounds that closely match the upper bounds, by using this approach on rebalancing cost at just the bottom level nodes.

Example 1: The bound of Theorem 1 for (u,v)-trees of hysteresis p is tight for arbitrary underflow context if v is odd and $u \geq \lceil (v+1)/3 \rceil$. I.e. for any initial tree T, there is an update sequence S for which $B(k,T,S) \geq k/p - o(k)$.

Construction: Let $m = u+p$, and observe that if an m-node x has 2 adjacent m-node brothers, then p+1 contractions and p+1 expansions of x will i) cause 1 merge and 1 split at x, ii) recreate 3 m-nodes, and iii) cause a contraction and expansion at x's father. We first show, for each $h>1$, an initial tree T^h of height h+1 and update sequence S^h for which $B(k,T^h,S^h) \geq k/p - k/(p+1)^h - O(h)$. Let T^h be the height h+1 tree with a binary root and two complete m-ary subtrees. Select a subgraph P^h of T^h, consisting of a complete height h (p+1)-ary tree plus the root of T^h, such that no two nodes of P^h are brothers or share a common brother in T^h. The update sequence S^h cyclically deletes the $(p+1)^h$ leaves of P^h from left to right, then reinserts them. Each cycle in S^h first merges all nodes of P^h in postorder, then splits them; thus the claimed bound on $B(k,T^h,S^h)$ follows easily.

To show $B(k,T,S) \geq k/p - o(k)$ for arbitrary initial T, we can construct S as a sequence of epochs S_i. Epoch S_i first constructs T^{i+1}, then performs enough updates in S^{i+1} to make $B(k,T,S_0 \ldots S_i) \geq k/p - k/2^i$.

We next show that for any (2,4)-tree algorithm, even with nondeterministic or clever splitting orientation and underflow context, there is always some update sequence on which the algorithm can never expand a 2-node, contract a 3- or 4-node, or share with a 4-node. This shows that the bound of Theorem 2 is tight.

Example 2: For any (2,4)-tree algorithm with merge threshold t=0 on initial tree T, $B(k,T,S) \geq 3k/2 - o(k)$ for some update sequence S.

Construction: Following Example 1, we give an initial tree T^h and sequence S^h for each $h>1$ with $B(k,T^h,S^h) \geq 3k/2 - k/2^h - O(2^h)$. Then a similar argument using epochs shows the result. Let T^1 be the height 1 3-ary tree, and T^{h+1} be a tree with a 3-ary root and sons (T^h, X^h, T^h), where X^h is an arbitrary height h (2,4)-tree. Sequence S^h repeatedly inserts and then deletes 2^h leaves in T^h, as follows. Insert 2^h leaves, 2 in each T^1 subtree- this splits the root of each T^i, $1 \leq i \leq h$, into a 2-node and a 3-node. At this point there is a unique path of 2-nodes from the root to a leaf, each with a 3-node brother. We arrange the deletions so that each is on a path consisting entirely of 2-nodes, forcing the underflow context to be unique.

The deletions under each 2-node x of height h+1 (with 2-node son y and 3-node son z) occur in the following order: i) (recursively) delete leaves from subtree y, until y underflows and borrows subtree X^h from z; ii) delete leaves from subtree z (now a 2-node) until z underflows; at this point z and y merge and x underflows. 2^h deletions reconstruct the initial tree, and the process repeats. The lower bound follows from the fact that every rebalancing operation except the split and merge at the root decreases account V (in Theorem 2) by exactly 2/3.

A more difficult construction (because of an algorithm's options for underflow context) shows that the bound of Theorem 2 is tight for all (u,2u)-trees [Hu81].

5. Subhistory Analysis

This section considers rebalancing cost in B-tree subhistories, bounded sequences of updates on some arbitrary initial tree. The preceding bounds on asymptotic cost can be improved for subhistories by two refinements in accounting: considering only the initial tree subset which influences rebalancing, and tuning savings accounts to adjust the magnitude of initial tree influence. The analysis can be used to identify, among algorithms with identical asymptotic behaviour, those with better subhistory behaviour.

Consider an arbitrary initial tree T_0 and a sequence S_k of k updates which creates tree T_k from T_0. Let $p_1,\ldots,p_k$ be the set of leaf positions in T_0 at which updates occur during S_k, and A be the set of ancestors of $p_1,\ldots,p_k$. Let M be A together with all adjacent brothers of nodes in A. Then A contains all "active" nodes in T_0, those which eventually overflow or underflow and initiate rebalancing during S_k, and M contains any other "passive" nodes that participate in rebalancing during S_k only as the brother of an underflow node. We derive a bound on the size of A and M below. We will then consider how savings accounts and knowledge about A and M can be used in analyzing rebalancing cost.

Theorem 6 bounds $|A|$ in any (u,v)-tree in terms of distinct positions $p_1,\ldots,p_{k'}$, $k'\leq k$. It generalizes a similar theorem in [BT80] for minimum 2-way branching to u-way branching. The proof employs a novel technique based on a local partition of the data structure. The technique gives an exact bound, and also circumvents the need to consider lowest common ancestors and telescoping sums in u-ary arithmetic.

Theorem 6: Let T be a level-complete multiway tree with $|T|$ leaves, a j-ary root, and minimum u-way branching except at the root. Let A be the set of ancestors of leaves at positions $p_1<\ldots<p_k$, and $d_i = p_{i+1}-p_i+1$. Let $j' = \min(j,2u)$. Then

$$|A| \leq 1-j'/(u-1) - k(2\log_u 2-1/(u-1)) + 2\sum_{1\leq i<k} \log_u d_i + 2\log_u(p_1+|T|-p_k+1),$$

and the bound is exact infinitely often.

Proof sketch: (Conceptually) mark all nodes in A. To count $|A|$, we will consider nodes as divisible quantities, areas, and analyze the marked area in the tree. Each (interior or leaf) node has area 1, and the root has area $R = \min(j/u,2)$. We also (conceptually) augment the tree with its "apex", fractional area ancestors of the root. The i'th root ancestor has area R/u^i, so the apex has total area $R/(u-1)$.

We define a disjoint partition of internal node and apex area into "slices". For each leaf x, the slice s_x contains a fraction of each ancestor of x. We also define an expanded slice S_x, used for the exact bound, which is a superset of s_x that intersects the i'th ancestor of x in area exactly $1/u^i$. We have $\text{area}(s_x) \le \text{area}(S_x) = 1/u+1/u^2+\ldots = 1/(u-1)$ for any leaf x.

Consider the marked area A_i in the expanded slice S_i spanning leaves p_i to p_{i+1}. The key observation is that the area of S_i decreases exponentially by level, and is < 2 at any level above $h = \lfloor \log_u(d_i/2) \rfloor$, independent of the height of the lowest common ancestor of p_i and p_{i+1}. We find

(1) $A_i \le f(d_i) = 2h + d_i u^{-h}/(u-1)$ and (2) $f(d_i) \le 2\log_u(d_i/2)+2/(u-1)$;

f is piecewise linear and continuous, and equality holds in (2) at the corners of f.

The bound on $|A|$ follows by summing the marked area of slices S_i, and correcting for root and apex area and the overlap of one leaf between adjacent slices. The bound is exact precisely when all expanded leaf slices are disjoint, and equality holds in (1) and (2); these can be translated into conditions on T and $p_1 \ldots p_k$.

A similar bound holds for the set M-A of brothers of A, and we also have the easier bound $|M-A| \le 2|A|-1$.

Now consider the savings account argument for a B-tree subhistory S_k. For a given (u,v)-tree algorithm and savings account V, let the rate of V be $1/w$, and the (active) capacity of V be $C = W/w$, where w is the minimum amount by which each rebalancing operation decreases V, and $W = \max_{u \le j \le v} V(j) \ge \max_{0 \le j \le v} V^R(j)$. Let M_k be the set of nodes in T_k that are affected by S_k, or that are in $M_0 = M$ but unchanged. Then $V(T_0)-V(T_k) = V(M_0)-V(M_k) \le V(M_0) \le W|M|$.

Thus the savings account argument on subhistory S_k gives the bound

(1) $B(k,T_0,S_k) \le k/w + C|M|$.

This could be a good bound for large k or clustered updates. But when $W|M| >> k$ for small or moderate k, one's intuition says (correctly for most algorithms) that for some constant c there is a better bound

(2) $B(k,T_0,S_k) \le ck + |A|$.

Theorem 7 gives a general bound on subhistory rebalancing cost, including both (1) and (2) and a spectrum of intermediate bounds, in terms of the parameters of a savings account.

Theorem 7: Suppose some (u,v)-tree algorithm performs a sequence S of k updates on an arbitrary initial tree T. Let M and A be as defined previously, and P = M-A. Let V be a savings account with rate w and active capacity C. Also let C' be the passive capacity for V, $0 \le C' \le C$, as defined in the proof. Then

$$B(k,T,S) \le k/w + C|A| + C'|P|.$$

Proof sketch: For each $i \le k$, we keep track of three sets affected by the first i updates S_i. A^i contains ancestors in $T_0 = T$ of the first i update positions in T_0 (formally defined by a leaf labelling process). P^i contains "passive" nodes in T_0 rebalanced during S_i, and is a subset of the adjacent brothers of A^i. M_i consists of successor nodes in T_i (including rebalanced nodes) of M^i, the union of A^i and P^i.

Consider M^i as a "reservoir" of node capacities that can be converted into rebalancing operations. It is immediate from the definition of W that $V(M_i) \le i + W|M^i| - wB(i,T_i,S_i)$. This is improved by observing that a passive node, after it is rebalanced, contributes at most W' to the reservoir, $0 \le W' \le W$. The passive capacity C' = W'/w is computed from the set of possible merge and share operations. Induction on i establishes the theorem by showing

$$V(M_i) \le i + W|A^i| + W'|P^i| - wB(i,T_i,S_i).$$

This theorem gives a spectrum of upper bounds on B(k,T,S) for an algorithm X, showing a tradeoff between initial tree influence and rebalancing rate attributed to updates by different savings accounts. We call the minimum (known) rate 1/w the asymptotic rate $1/w_a$ for algorithm X. For some algorithms, any account V which realizes rate $1/w_a$ requires capacity $C >> 1$. We call the minimum rate for an account V which realizes minimum capacities (C=1 and C'=0 for most algorithms) the locally robust rate $1/w_c$ for algorithm X.

We will show how to construct savings accounts for the locally robust rate. For any (u,v)-tree algorithm in $Alg(t,s_{min})$ with $v > 2u$ and $t < v-2u+1$, we give an account V with capacities C=1 and C'=0. Consider an x-node and a y-node created by splitting, or a z-node created by any merge operation. Let $w_1 = \min(x,y)-u$, $w_2 = v-z$, and $w_3 = \min(s_{min},(t+1)/2)$. Let $w_c = \min(w_1,w_2,w_3)$, the locally robust rate, and define V (constructed to satisfy V(x)=V(y)=V(z)=0) by

$$V^I(j) = \max(0, (u+w_c)-j, j-(v-w_c)).$$

Thus w_c=1 for Alg(2,1), 3/2 for Alg(2,2), and 2 for Alg(3,2) on (u-2,2u)-trees.

We can use subhistory analysis to choose, among algorithms with identical asymptotic rates, those with better subhistory behaviour. For example, any (40,99)-tree algorithm with merge threshold $t \ge 14$ and shift hysteresis $s_{min} \ge 5$ has optimal asymptotic rate 1/10, and $B(k) \le k/10 + 8.8|M| \le k/10 + 26.4|A| - 1$. If t = 14 we also have $w_c = 5$, hence $B(k) \le k/5 + |A|$. But w_c is only 1 for t = 19, and we haven't defined w_c for t = 20. In fact, simple examples with $|A| >> k$ for bounded but large k demonstrate $B(k) \ge k + |A|-O(1)$ for t=19 and $B(k) \ge 2|A|-1$ for

t=20, showing that t = 15 is the preferred choice. Consideration of minimal capacity accounts for asymptotic or intermediate rates permits similar discriminations among sharing strategies in asymptotically equivalent algorithms.

6. B^*-trees

B^*-trees are a generalization of B-trees motivated by improved space utilization, first suggested in [BM72], and further developed and named in [Kn73]. B^*-trees generalize B-tree algorithms by allowing larger neighborhoods of nodes to be used in rebalancing. Let BSTAR be the standard B^*-tree algorithm in [Kn73], which maintains a (2u+1,3u+1)-tree data structure. It balances nodes by sharing with an adjacent brother if possible, and uses 2-way splitting (2 nodes into 3) or 3-way merging (3 nodes into 2) otherwise. We will show that robust balancing also works for B^*-trees, and consider some of the implications of this fact.

Theorem 8: Consider algorithm BSTAR on an initial (u,v)-tree T. Then

a) $B(k) \geq \Theta(k \log|T|)$ in the worst case if $3(u-1)=2(v-1)$,

b) $B(k) \leq 2k + O(|T|)$ if $3(u-1) < 2(v-1)$.

Proof sketch: a) Let T be a complete u-ary tree, and repeatedly delete and reinsert the leftmost leaf.

b) Suppose that algorithm BSTAR still rebalances by sharing whenever possible, although $3(u-1)<2(v-1)$ makes other choices available. Let V be the savings account defined on interior nodes by

$V^I[u-1,u,u+1,\ldots,v-2,v-1,v,v+1] = [3/2,1/2,0,\ldots,0,1/4,3/4,7/4]$.

Then each rebalancing operation decreases V by at least 1/2, and the theorem follows by the savings account argument.

We can trade a large decrease in worst case rebalancing cost for a small increase in space usage in B^*-trees as well as in B-trees. Let the B^*-robustness r of (u,v)-trees be $r = 2(v-1)-3(u-1)$, and consider variations BSTAR(t,s_{min}) with parameters analogous to those for B-trees. Then by properly choosing t and s_{min} for BSTAR(t,s_{min}) on (u,v)-trees of robustness r, we have $B(k) \leq ck/r + O(|T|)$ for some constant c and initial tree T. Optimal bounds are difficult to establish for B^*-trees, but locally robust accounts as in Section 5 suffice to show this result.

B^*-trees demonstrate that robust balancing is significant for monotone insertions as well as for intermixed insertions and deletions, since parameters t and s_{min} apply to overflow as well as underflow. The robust balancing technique of selecting $s_{min} > 1$ prevents the problem of thrashing by share operations during insertions (one sharing per update) noted in [BM72].

Worst case space utilization for (u,v)-tree algorithms is determined by the degree of merging, 2-way for B-trees and 3-way for algorithm BSTAR. Other B^*-tree algorithms can achieve robust balancing with no decrease in efficiency of space

utilization by using the same degree of splitting as merging. For example, an algorithm with 2-way splitting and merging can maintain (u,2u-1)-trees with a linear rebalancing cost, and a sublinear rebalancing cost when only insertions occur.

7. Conclusion

Robust balancing gives an order of magnitude improvement in worst case cumulative rebalancing cost over the standard balancing method. When used with fingers, it can decrease total update cost to linear time.

We develop an accounting technique that gives tight or nearly tight bounds on rebalancing cost, with elegant accounts that show the asymptotic behaviour.

Optimal algorithms for robust B-trees achieve sublinear total rebalancing cost at only a small cost in space utilization. This is significant for B-trees stored on disk, and for applications with expensive balancing operations.

The results for B^*-trees give additional importance to robust balancing in practice, and nullify most objections to its use. If one objects to robust balancing in B-trees on grounds of space utilization, then it is better to use B^*-trees with robust balancing than B-trees with fragile balancing. The potential for thrashing by share operations on insertions in B^*-trees also demonstrates that robust balancing is significant for monotone insertions as well as for intermixed insertions and deletions.

References

[BM72] Bayer, R. and E. McCreight: Organization and Maintenance of Large Ordered Indexes. Acta Informatica 1, 173-189 (1972).

[BS77] Bayer, R. and M. Schkolnik: Concurrency of Operations on B-trees. Acta Informatica 9, 1-21 (1977).

[BT80] Brown, M.R. and R.E. Tarjan: Design and Analysis of a Data Structure for Representing Sorted Lists. SIAM J. on Computing 9, 594-614 (1980).

[Kn73] Knuth, D.E.: The Art of Computer Programming: vol. 3. Addison-Wesley. 1973.

[Hu79] Huddleston, S.: The Average Rebalancing Cost in 2-3-4 Trees is Constant. unpublished note. 1979.

[Hu81] Huddleston, S.: Robust Balancing in B-Trees. PhD Dissertation, Computer Science Dept., University of Washington, Seattle. 1981.

[HM80] Huddleston, S. and K. Mehlhorn: A New Data Structure for Representing Sorted Lists. Fachbereich 10 - Informatik, Universität des Saarlandes, Saarbrücken. 1980.

[MS79] Maier, D. and S.C. Salveter: Hysterical B-Trees. Tech. Rpt. No. 79/007, Dept. of Computer Science, State Univ. of New York at Stony Brook. 1979.

[Wi79] Willard, D.E.: The Super-B-tree Algorithm. Harvard Aiken Computation Lab. Report TR-03-79. 1979.