NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:
The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.

Competitive Algorithms for Replication and Migration Problems

David L. Black and Daniel D. Sleator

November 1989
CMU-CS-89-2013

Abstract

In this paper we consider problems that arise in a shared memory multiprocessor in which memory is physically distributed among a number of memories local to each processor or cluster of processors. The issue we address is that of deciding which local memories should contain copies of pages of data. In the migration problem we operate under the constraint that a page must be kept in exactly one local memory. In the replication problem we allow a page to be kept in any subset of the local memories, but do not allow a local memory to drop a page once it has it.

For interconnection topologies that are complete graphs, or trees we have obtained efficient *on-line* algorithms for these problems. Our migration algorithms also extend to interconnections that are products of these topologies (e.g. a hypercube is a product of simple trees). An on-line algorithm decides how to process each request (which is a read or write request from a processor to a page) without knowing future requests. Our algorithms are also said to be *competitive* because their performance is within a small constant factor of that of any other algorithm, including algorithms that make use of knowledge of future requests.

This research was supported in part by the National Science Foundation under grant CCR-8658139.

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), monitored by the Space and Naval Warfare Systems Command under Contract N00039-87-C-0251.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

1. Introduction

A common design for a large shared memory multiprocessor system is a network of processors for which each processor or cluster of processors has its own local memory [3,13,15]. In such a design, a virtual memory system supports a programming abstraction of memory as a single address space without restrictions on how the pages of this address space are distributed among the local memories. A page of this abstract memory may be stored in just one local memory, or identical copies of it may be replicated in many local memories. When a processor p needs to read a location l in a page b, it first looks to see if b is in its own local memory. If it is, the access is accomplished locally. If it is not, a request is transmitted over the network to a local memory containing the desired page, and the contents of l are transmitted back to p.

To clarify the trade-offs involved it is helpful to consider two extreme cases. Suppose that a page b is read very often by all of the processors, but is never written. In this case, the network use is minimized by replicating b in all of the caches. Once this initial cost is incurred, no further network communication is needed. On the other hand, if a page b is repeatedly written by one processor p, then it behooves the system to eventually migrate b to p's cache, after which no network communication is needed. Since communication bandwidth is frequently a bottleneck in such architectures, we have focused or attention on finding residency strategies that attempt to reduce the total interconnection bandwidth used in processing a sequence of requests.

Most multiprocessors do not have broadcast, invalidate, or snooping mechanisms that maintain consistency among multiple copies of a page when writes occur (note that we are considering multiple memory copies, as distinct from multiple cache copies which can be kept consistent). As a result we must restrict writeable pages to a single copy, so the residency problem becomes one of deciding which local memory should contain this copy; we call this the *migration* problem. The corresponding problem for a read-only page is to determine which of the local memories should contain copies of the page. We call this the *replication* problem because this set of local memories that contains copies of the page is monotonically non-decreasing. We separate the issue of reclaiming memory used in this fashion because there are other competing uses for this memory; a companion paper covers this and related issues in more detail [2]. If a page is both read-only and writable at different times, we consider each segment (read-only or read/write) of the page's existence to be a separate instance of the corresponding problem.

Karlin et.al.[10] considered a related problem of cache residency on bus-based multiprocessors with coherent caches. These problems are generalizations of the ones we consider because multiple copies of writable data are allowed to exist (and are kept consistent), but the authors only considered bus-based interconnections. This corresponds to a network in which the distance between any pair of nodes is the same, that is, a complete graph with uniform distances. They called this problem *general snoopy caching*, and obtained an algorithm for this problem whose performance is within a factor of three of that of any algorithm for any sequence of requests.

An algorithm is said to be *on-line* if makes a decision about how to process a request based only on that request and the ones before it. Karlin et.al. describe a general framework in which to analyze on-line algorithms. They called an on-line algorithm A *c-competitive* if there exists a constant a such that for every sequence of requests σ, and every algorithm B (on-line or off) we have:

$$C_A(\sigma) \leq c \cdot C_B(\sigma) + a,$$

where $C_A(\sigma)$ is the cost incurred by algorithm A on σ, and $C_B(\sigma)$ is defined analogously.

The migration problem can now be stated simply, without reference to the motivating multiprocessor. We're given a network (a graph in which each edge has a length). At any time, there is exactly one node of the network that is special, this is the node that has the page. A sequence of accesses to the page are generated at the nodes of the network. The cost of an access is the distance between the accessing node and the node with the page. In addition to paying for a request, an algorithm is also allowed at any time to move the page from where it is to another node. The cost of a move is m times the distance between the starting and destination nodes, where m is a constant (which roughly corresponds to the size of the page).

In this paper we consider on-line algorithms for this problem with look-ahead zero. Such an algorithm must satisfy a new request in the current state, and after satisfying the request it is allowed to change its state by moving the page. (This is in contrast to a look-ahead one algorithm, which would be allowed see the request, move the page, and finally satisfy the request. Look-ahead zero is more natural in this setting. Section 4 discusses look-ahead one versions of our algorithms.)

The networks that we consider are those in which the distances are symmetric and satisfy the triangle inequality. If the actual physical network being analyzed does not have a link between every pair of nodes, then the distance between them is the shortest path length in the network between the two nodes.

We have obtained 3-competitive algorithms for the migration problem on the complete network with uniform distances, on any network whose distance metric is that of a tree, and on any network that is the product of trees and/or complete graphs (e.g. a hypercube). A result obtained by Karlin et.al shows that these algorithms are *strongly competitive* in the sense that 3 is the minimum possible competitive factor.

The migration problem is closely related to the problem of 1-server with excursions, defined (but not studied) by Manasse et.al.[12]. The migration problem is a special case of 1-server with excursions obtained by restricting the cost of a move (migration) to a uniform constant (m) times the cost of a remote access. In contrast, the move cost for the general 1-server with excursions problem can be arbitrary. Based on our results we conjecture that 3-competitive on-line algorithms exist for the migration problem on all topologies with symmetric distance metrics that satisfy the triangle inequality.

Section 3 contains our results on migration. We describe algorithm M, a 3-competitive algorithm for migration on a network of uniform distances and algorithm M-Tree, a 3-competitive algorithm for the migration problem on any network with the distance metric of a tree. We also consider a specialized version of algorithm M-Tree (called M-UTree) for networks whose distance metric is that of a tree with edges of length 1; the algorithm uses fewer counters than M-Tree, but is only 4-competitive.

We analyze our replication algorithms slightly differently from our migration algorithms. Strictly speaking, the trivial algorithm that initially replicates the page to all of the nodes is 0-competitive for the problem, since we can put all of the cost of the initial page movement into the constant a in the definition of competitiveness. To give meaningful results for this problem we therefore redefine c-competitiveness to mean satisfying the above inequality with an additive constant of zero.

Section 2 contains our results on replication. We give 2-competitive algorithms for the replication problem on uniform networks, and networks with the distance metric of a tree. We also give a specialized version for trees in which each edge has length one that uses less state than the general tree algorithm. These algorithms are strongly competitive, as it is fairly easy to show that the lower bound on the competitive factor for this problem is 2.

In Section 5 we describe in more detail how our idealized models for migration and replication relate to real multiprocessor systems.

We consider two topologies for our algorithms:

Complete A complete graph in which every pair of distinct nodes are separated by the same distance.

Tree A tree in which distance is additive (i.e. there is a unique path between any two nodes and the access cost is the sum of the access costs for the individual edges along that path).

We also consider a variant of **Tree**, **UTree** in which all single edge access costs are identical (i.e. the cost of an access is the number of edges multiplied by a fixed constant d); simpler algorithms exist given this cost assumption. Our migration algorithms also extend to products of these graphs by employing an independent instance of the appropriate algorithm in each dimension of the product graph; important examples of such products are hypercubes and meshes, which are products of linear trees.

2. Replication

We can now give the general abstract form of the replication problem. A set of n nodes, and a distance metric δ_{ij} that specifies the distance between all pairs of nodes i and j are given. In this paper we shall only be concerned with distance metrics that are symmetric and satisfy the triangle inequality. A graph G with lengths on its edges is said to satisfy the distance metric δ_{ij} if the shortest path in G between i and j is δ_{ij}.

The general state of the system is described by a bit vector with one bit per node. A node whose bit is 1 is said to 'have the page'. In the initial state of the system there is a particular single node (which we shall always call s) that has the page. As time goes on the bits of other nodes change to 1. When this happens to a node, it is said to have 'replicated the page'. Once a node has a copy of the page, it retains it.

A sequence of requests to nodes is to be satisfied. The cost of satisfying a request is the distance from the requested node to the nearest node with the page. The cost of replicating the page is r times the distance to the nearest node with the page.

The replication problem is to decide (in an on-line fashion) which nodes should have the page, and to do this in a way that has low cost. As usual we shall compare the performance of a prospective on-line algorithm to that of the off-line optimum on the same sequence of requests. We shall assume that the on- and off-line algorithms start in the same state, the one in which the page is in node s.

It is easy to see that no on-line algorithm (for any non-trivial version of this problem) can achieve a cost that is less than twice the optimum on all sequence of requests. Let s and t be the two nodes. Consider the sequence of requests that accesses t repeatedly until the on-line algorithm has given node t the page. Let k be the number of requests in this sequence. The cost incurred by the on-line algorithm is $(k+r)\delta_{st}$. If $k \geq r$ then the optimum off-line algorithm replicates the page immediately and incurs a cost of $r\delta_{st}$. If $k < r$ then the optimum off-line algorithm never replicates the page and incurs a cost of $k\delta_{st}$. In either case the off-line algorithm incurs a cost that is at most half of the on-line algorithm.

Our goal is thus to find on-line algorithms that achieve this factor of two for various distance metrics. We have done this for the cases in which the metric is that of a tree, and the case in which all distances are equal.

It is possible to prove that for certain other metrics (for example, when the graph corresponding to the metric is a four-node cycle) the best competitive factor that an on-line algorithm can hope for is $5/2$. We leave these questions to future research.

2.1. Replication for two nodes

There is a very simple algorithm to achieve the factor of two when there are just two nodes. It is easy to derive the algorithm (which we call C) from the lower bound proven above. Let s and t be the two nodes. Algorithm C keeps a count of the number of requests to t. When this reaches r the page is replicated into t. Like the lower bound, the proof that this algorithm is within a factor of two of optimum breaks into two cases. Let k be the number of requests in the sequence. The cost incurred by algorithm C is $k\delta_{st}$ if $k < r$ and $2r\delta_{st}$ otherwise. In the former case the algorithm's performance is optimum, in the latter its performance is within a factor of two of the minimum cost, $r\delta_{st}$.

2.2. Replication for the Uniform Problem

It is very easy to generalize algorithm C to many nodes in the case in which the distance between every pair of nodes is the same.

> **Algorithm R:** Maintain a count on each node other than s. The count on a node is incremented each time that node is accessed. When the count on a node i reaches r, the page is replicated into i.

To analyze this algorithm we partition the cost of a sequence of requests into the costs incurred by the requests to each vertex. The costs incurred by a vertex include the cost of the accesses to that vertex, and the cost of replicating the page there. Each vertex represents an entirely separate two vertex problem. Algorithm **R** is merely running algorithm **C** on each of these separate problems. Therefore its performance is within a factor of two of optimum.

This analysis is also applicable to *star shaped graphs*, which are graphs with a central node s such that the distance from any node i to any other node j is not less than the distance from i to s. Hence Algorithm **R** is also within a factor of two of optimum for such graphs.

2.3. Replication for Trees

Another easy generalization of algorithm C is to the case in which the distance metric is a tree.

> **Algorithm R-Tree:** The algorithm maintains a count (initially zero) on every node. When a node i that does not have the page is accessed, the count of every node along the path from i to the closest node with the page is incremented. The page is replicated to all nodes whose counts reach r after the access. (Of course it is not necessary to maintain counts on nodes that have the page. We have expressed the algorithm this way to simplify the following exposition.)

This algorithm is also within a factor of two of the optimum off-line algorithm. Before we can prove this we need the following observation: The counts on the nodes of a path from s to any other vertex are monotonically non-increasing. This fact is initially true, and is easy to prove by induction. It is also easy to prove that after an access the nodes with the page are exactly those with counts of r or more.

Consider any algorithm A (off-line or on-line) for this problem. We can assume without loss of generality that if algorithm A arranges things so that node i has the page, then all the nodes on the path from s to i also have the page. We can make this assumption because if the algorithm does not do this replication, then it can be modified so that it does the replication, and incurs no more cost. Thus for any algorithm A we can assume that the nodes with the page are a connected component in the tree.

These constraints allow us to analyze algorithm **R-Tree** by partitioning the costs incurred by it and by A into parts corresponding to the edges of the tree. An edge incurs a cost for an access operation (equal to the length of the edge) if the path from the accessed node to the closest node that has the page passes through the edge. Otherwise the cost incurred by the edge is zero. The edge also incurs the cost of a replication across it.

We can view the behavior of algorithm **R-Tree** and any other algorithm A from the perspective of a particular edge. With respect to the game being played on this edge, algorithm **R-Tree** is doing exactly what algorithm C would do: when the count on the end without the page reaches r the page is replicated across the edge. The total cost incurred by an algorithm is just the sum of the costs incurred by all the edges. For each edge algorithm **R-Tree** is within a factor of two of the cost of any other algorithm for that edge. This proves that **R-Tree** is within a factor of two of optimum.

2.4. Replication for Uniform Trees

One disadvantage of algorithm **R-Tree** is that it must keep state for every node in the tree, even though a page can only be replicated to adjacent nodes. If the single edge distances in the tree are constant, this state can be collapsed. The resulting algorithm still involves counters for each node, but the counters for nodes that do not have the page and are not adjacent to copies are always zero. This means that counters need only be maintained for nodes that are adjacent to copies of the page. We will call these nodes *boundary nodes*. Since we are starting with exactly one copy of the page, there is always a unique closest boundary node to any non-boundary node that does not have the page. Our algorithm to implement replication using *boundary nodes* is:

> **Algorithm R-UTree:** Initialize the counters c_i to zero. The algorithm processes a request from a node that does not have the page as follows: find the path to the closest copy of the page, and add the length of the path to the counter for the boundary node on the path. If that counter is $\geq r$, replicate the page into the boundary node, and zero that node's counter. If the value before replication was $> r$ set the counter for the new boundary node on the path to the original boundary node's counter value less r; if this value is $\geq r$, the algorithm loops back to replicate the page into the new boundary node. If the request originated at the original boundary node, then the original counter was $r - 1$ before the request, and there is no excess value to be assigned.

The following theorem establishes that algorithm **R-UTree** is strongly competitive.

Theorem 1 *For any sequence σ of requests for the tree page replication problem with constant single edge access costs and any on-line or off-line algorithm $\mathcal{A}$*

$$C_{\text{R-UTree}}(\sigma) \leq 2 \cdot C_{\mathcal{A}}(\sigma)$$

under the assumption that $\mathcal{A}$ and **R-UTree** *start in the same state with a single copy of each page.*

Proof: Assume without loss of generality that all single edge distances in the tree are 1. We merge the actions taken by the two algorithms into a single sequence of events tagged to indicate the algorithms involved. We shall give a non-negative (initially zero) potential Φ such that the following inequality is satisfied by every event:

$$2 \cdot \Delta C_{\mathcal{A}} - \Delta C_{\textbf{R-UTree}} \geq \Delta\Phi(t)$$

where the Δ indicates the change in the value of the parameter as a result of the event. Summing this formula over all events and using the fact that the initial potential is no more than the final potential yields the theorem. It remains to specify the potential and verify the above inequality.

Let S be the set of nodes i such that only $\mathcal{A}$ has a copy of the page in node i. We define the potential function as:

$$\Phi(t) = \sum_{i \notin S} c_i + \sum_{i \in S} (2r - c_i)$$

Every step in either algorithm that changes the potential or incurs a cost results in an event. We now proceed to establish the desired inequality for all possible events.

Consider a replication action performed by $\mathcal{A}$. Let i and j be the source and destination nodes. Since i and j are adjacent, $\delta_{ij} = 1$ by assumption. The cost of the replication to $\mathcal{A}$ is r, so we must show that $\Delta\Phi \leq 2r$. There are two cases to consider based on whether j belongs to S after the replication:

$j \in S$: This means that j does not have a copy of the page under **R-UTree**. j is added to S, so $\Delta\Phi = (2r - c_j) - c_j = 2r - 2c_j \leq 2r$ because $c_j \geq 0$.

$j \notin S$: This means that j already has a copy of the page under **R-UTree**. There is no change to S so $\Delta\Phi = 0$.

Consider a replication action performed by **R-UTree**. This action must be analyzed in combination with the pair of actions that satisfy the request for both algorithms. Let i and j be the source and destination nodes. There are a total of five cases depending on whether j and e are members of S before the replication and the distance of the access. Let d be the path length of the request; the two simplest cases are for $d = 1$. In this case, $c_j = r - 1$ before the replication, and $c_j = 0$ afterwards. $\Delta C_{\textbf{R-UTree}} = r + 1$ to account for the replication and initial access. The two $d = 1$ cases are:

$j \notin S$: $\Delta C_{\mathcal{A}} = 1$, so we must show $\Delta\Phi \leq 2(1) - (r+1) = 1 - r$. $\Delta\Phi = c_j' - c_j = 0 - (r-1) = 1 - r$.

$j \in S$: This case is free to $\mathcal{A}$, so we must show $\Delta\Phi \leq 2(0) - (r+1) = -(r+1)$. $j \notin S$ after the replication, so $\Delta\Phi = c_j' - (2r - c_j) = 0 - (2r - (r-1)) = -(r+1)$.

$d > 1$ for the remaining three cases. Let e be the new boundary node for the request that caused the replication (after the replication). $c_j = r - x$ and $c_e = 0$ before the replication where $1 \leq x \leq d$. $c_j' = 0$ and $c_e' = d - x$ after the replication. $\Delta C_{\textbf{R-UTree}} = r + d$. There are three remaining cases depending on whether j and e belong to S before the replication.

$j, e \notin S$: $\mathcal{A}$ has not replicated the page beyond i, so $\Delta C_{\mathcal{A}} \geq d$, and we must show $\Delta\Phi \leq 2d-(r+d) = d-r$. But $\Delta\Phi = (c_j'+c_e')-(c_j+c_e) = 0+(d-x)-(r-x)-0 = d-r$.

$j \in S,\ e \notin S$: $\mathcal{A}$ has replicated the page to j, but not beyond, so $\Delta C_{\mathcal{A}} = d-1$, and we must show $\Delta\Phi \leq 2(d-1)-(r+d) = d-r-2$. $\Delta\Phi = (c_j'+c_e')-((2r-c_j)+c_e) = 0+(d-x)-(2r-(r-x))-0 = d-x-2r+r-x = d-r-2x$ Hence $\Delta\Phi \leq d-r-2$ because $x \geq 1$.

$j, e \in S$: $\mathcal{A}$ has replicated the page beyond j, so $\Delta C_{\mathcal{A}} = d-k$ where $2 \leq k \leq d$, and we must show $\Delta\Phi \leq 2(d-k)-(r+d) = d-r-2k$. $\Delta\Phi = (c_j'+(2r-c_e'))-((2r-c_j)+(2r-c_e)) = 0+(2r-(d-x))-(2r-(r-x))-(2r-0) = 2r-d+x-2r+r-x-2r = -r-d$. Since $k \leq d$ we have $\Delta\Phi = -r-d = d-r-2d \leq d-r-2k$ as was to be shown.

A request whose length is greater than r may cause more than one replication; these replications occur in sequence, and the above analysis applies by setting the request length (d) for subsequent replications to the previous length less the amount required to cause the previous replication.

The remaining actions involve satisfying the request. We pair off the corresponding local and/or remote supply actions for a single request and deal with them as a pair provided that they were not dealt with in the previous case. Let r and s respectively be the nodes from which $\mathcal{A}$ and **R-UTree** supply the location, and let t be the node to which it is supplied. When needed, let e be the boundary node for **R-UTree** and this request. Then there are three cases to consider:

$r = s$: If $r = s = t$ then there are no cost or potential changes. Otherwise both algorithms incur costs of δ_{st}, so we must show that $\Delta\Phi \leq \delta_{st}$. c_e increases by δ_{st}. Because both algorithms performed a remote supply, $e \notin S$, so $\Delta\Phi = \delta_{st}$.

$r \neq s,\ \delta_{rt} > \delta_{st}$: $\mathcal{A}$ incurs a cost of δ_{rt}, **R-UTree** incurs a cost of δ_{st}, so we must show that $\Delta\Phi \leq 2\delta_{rt} - \delta_{st}$. But $\Delta\Phi = \delta_{st}$ as in the previous case, so $\Delta\Phi = \delta_{st} = 2\delta_{st} - \delta_{st} < 2\delta_{rt} - \delta_{st}$.

$r \neq s,\ \delta_{rt} < \delta_{st}$: $\mathcal{A}$ incurs a cost of δ_{rt}, **R-UTree** incurs a cost of δ_{st}, so we must show that $\Delta\Phi \leq 2\delta_{rt} - \delta_{st} = \delta_{rt} - \delta_{rs}$. c_e increases by δ_{st}. Because the distance for **R-UTree** (δ_{st}) is larger than the distance for $\mathcal{A}$, it follows that $e \in S$. Hence $\Delta\Phi = -\delta_{st} \leq 2\delta_{rt} - \delta_{st}$.

QED

3. Migration

In the migration problem, we must maintain exactly one copy of the page in the network, and we must decide on-line where to keep it. As in the replication problem let the cost of satisfying a request from a node that does not have the page be the distance between the requesting node and the node with the page in the network (denoted δ_{ij}). The cost of moving the page from i to j is given by $m\delta_{ij}$.

Before presenting our algorithms, we show that three is the best competitive factor that can be achieved for this problem.

3.1. Lower Bound

This section presents our result that the best possible competitive factor for any non-trivial instance (2 or more nodes) of the migration problem is 3. This situation surrounding this result is somewhat unusual because it has been proved in a previous paper [10], although it is not stated there. The reason for this is that the proof of one of the theorems in that paper establishes a more general result than claimed in the statement of the theorem.

The theorem in question is Theorem 3.3, which establishes a lower bound of 3 on the competitive factor for the "on-line block retention problem in a model allowing *Supplythrough* and *Updatethrough*" provided that there are "at least two caches." This problem differs from our migration problem in that the page (cache block) is permitted to be in more than one place at once, and there is a cost to satisfy certain requests (WRITE) locally if the page (cache block) exists in more than one place; these costs represent the overhead of maintaining cache consistency.

The theorem is proved by considering two caches and a single cache block. There are four possible states for the block; in neither cache, unique to the first cache, unique to the second cache, and in both caches. For an arbitrary algorithm A, the proof constructs a sequence σ consisting solely of WRITE requests for the cache block. This sequence is a "worst-possible" sequence for A because every access is costly. An off-line algorithm H is then described which uses lookahead to process σ more efficiently than A. The properties of σ and the design of H permit it to be shown that H's total cost is one third that of A's at infinitely many points in the infinite sequence σ.

The result can be generalized because H only uses the two states in which the block is unique to a single cache. The two node migration problem can be obtained from the two node block retention problem by restricting the class of algorithms to those which always keep the block present in exactly one cache (i.e. the page is always located at exactly one node). This restricts the selection of A to a subclass of the original algorithms. H is in this subclass (it only uses the two states in which the block is unique to a single cache), and hence the theorem holds for the subclass. This establishes the factor of 3 lower bound on the competitive factor for the migration problem. The theorem can be formally stated as:

Theorem 2 *Let $\mathcal{A}$ be any on-line page migration algorithm for a topology with at least two nodes. Then there is an infinite sequence of requests σ such that $\mathrm{C}_{\mathcal{A}}(\sigma(n)) \geq n$, and*

$$\mathrm{C}_{\mathcal{A}}(\sigma(n)) \geq 3 \cdot \mathrm{C}_{\mathbf{opt}}(\sigma(n))$$

for infinitely many values of n, where $\sigma(n)$ denotes the first n requests of σ.

Karlin et.al.[10] also proves that there is no best on-line algorithm for this caching problem; a similar theorem can be proved for the migration problem.

3.2. Migration on a Complete Graph

Algorithm **M** below solves the migration problem on a graph in which the access cost between any pair of nodes is one, and the move cost between any pair of nodes is an integer m. The algorithm maintains an integral count on each node. These counts are initially zero, and always lie in the range $[0, 2m]$. Let c_i denote the count on node i.

> **Algorithm M:** Initialize all counts c_i to zero. The algorithm processes a request to vertex v as follows: If vertex v has the page, then the request is free and nothing happens. If vertex v does not have the page and $c_v < 2m$ then increment c_v and decrement some other non-zero count if there is one. If vertex v does not have the page and $c_v = 2m$ then move the page to vertex v and set c_v to zero.

The following lemma establishes an important invariant satisfied by algorithm **M**.

Lemma 1 $\sum_i c_i \leq 2m$ *after the completion of each operation.*

Proof: By induction. All the counts are initially zero, so the sum is also. An operation that increments a counter increments the total sum only if all other counters are zero (else a non-zero counter is decremented, and the sum is unchanged). Attempting to increment a counter whose value is $2m$ resets it (and therefore the sum) to zero. Hence the sum must be $\leq 2m$ after each operation. QED

This Lemma has three important corollaries:

1. All counter values are bounded by 0 and $2m$.
2. Before the page is moved, the counter in the destination vertex is $2m$, and all other counters are zero.
3. After the page is moved, all of the counters are zero.

The following theorem establishes that algorithm **M** is strongly competitive.

Theorem 3 *Algorithm* **M** *is strongly 3-competitive for the migration problem. In particular, for any sequence σ of requests and any on-line or off-line algorithm $\mathcal{A}$*

$$C_{\mathbf{M}}(\sigma) \leq 3 \cdot C_{\mathcal{A}}(\sigma)$$

under the assumption that $\mathcal{A}$ and **M** *start in the same state.*

Proof: In analyzing the performance of these algorithms during a sequence of requests, we can partition the things that happen into a sequence of *events* of three types: algorithm **M** moves the page, algorithm $\mathcal{A}$ moves the page, and a request is satisfied by both algorithms. We shall give a non-negative (initially zero) potential function Φ such that the following inequality is satisfied for every type of event:

$$3 \cdot \Delta C_{\mathcal{A}} - \Delta C_{\mathrm{M}} \geq \Delta\Phi$$

The Δ indicates the change in the value of the parameter as a result of the event. Summing this formula over all the events, and using the fact that the initial potential is no more than the final potential gives the theorem. It remains to verify the above inequality.

Let the location of **M**'s page be s, and the location of $\mathcal{A}$'s page by t. The potential function we shall use is:

$$\Phi = \begin{cases} 2\sum_i c_i & \text{if } s = t \\ 3m - c_t + \frac{1}{2}\sum_{i \neq t} c_i & \text{if } s \neq t \end{cases}$$

Consider the event in which algorithm **M** moves the page from s to s'. The cost to **M** is m, and the cost to $\mathcal{A}$ is 0, so we need to show that $\Delta\Phi \leq -m$. The corollaries above simplify the calculation of $\Delta\Phi$. There are three cases:

$s' = t$: $\Delta\Phi = 2\sum 0 - (3m - 2m + \frac{1}{2}\sum 0) = -m$.

$s = t$: $\Delta\Phi = (3m - 0 + \frac{1}{2}\sum 0) - 2(2m) = -m$.

$s, s' \neq t$: $\Delta\Phi = (3m - 0 + \frac{1}{2}\sum 0) - (3m - 0 + \frac{1}{2}(2m)) = -m$.

Consider the event in which algorithm $\mathcal{A}$ moves the page from t to t'. The cost to **M** is 0, and the cost to $\mathcal{A}$ is m, so we need to show that $\Delta\Phi \leq 3m$. Again there are three cases:

$t' = s$: $\Delta\Phi = 2\sum_i c_i - \left(3m - c_t + \frac{1}{2}\sum_{i \neq t} c_i\right) = 3c_t + \frac{3}{2}\sum_{i \neq t} c_i - 3m \leq 6m - 3m \leq 3m$ because the counts are bounded by 0 and $2m$.

$t = s$: $\Delta\Phi = \left(3m - c_{t'} + \frac{1}{2}\sum_{i \neq t'} c_i\right) - 2\sum_i c_i = 3m - 3c_{t'} - \frac{3}{2}\sum_{i \neq t'} c_i \leq 3m$ because the counts are non-negative.

$t, t' \neq s$: $\Delta\Phi = \left(3m - c_{t'} + \frac{1}{2}\sum_{i \neq t'} c_i\right) - \left(3m - c_t + \frac{1}{2}\sum_{i \neq t} c_i\right) = \frac{3}{2}(c_t - c_{t'}) \leq 3m$ because the counts are bounded by 0 and $2m$.

Consider an event that is an access operation. Let r be the requested vertex. If $s = t$ there are two cases:

$r = s = t$: There is no cost, and no change to Φ.

$r \neq s = t$: Both algorithms incur a cost of 1, so we must show $\Delta\Phi \leq 2$. The counter increment always adds 2 to Φ. If another counter is decremented, 2 is subtracted from Φ, so $\Delta\Phi \in \{0, 2\}$.

If $s \neq t$ there are three cases:

- $r = s$: The cost to $\mathcal{A}$ is 1 and the cost to **M** is 0, so we must show $\Delta\Phi \leq 3$. If no decrement occurs, $\Delta\Phi = 0$. If the counter c_t is decremented $\Delta\Phi = 1$. Otherwise some other counter is decremented, and $\Delta\Phi = -\frac{1}{2}$.
- $r = t$: The cost to $\mathcal{A}$ is 0 and the cost to **M** is 1, so we must show $\Delta\Phi \leq -1$. The counter increment always subtracts 1 from Φ. If another counter is decremented then $\Delta\Phi = -\frac{3}{2}$. Otherwise $\Delta\Phi = -1$.
- $r \neq s, t$: The cost to $\mathcal{A}$ is 1 and the cost to **M** is 1, so we must show $\Delta\Phi \leq 2$. The counter increment always adds $\frac{1}{2}$ to Φ. If no decrement occurs then $\Delta\Phi = \frac{1}{2}$. If c_t is decremented then $\Delta\Phi = \frac{3}{2}$. Else some other counter is decremented, and $\Delta\Phi = 0$.

This completes the case analysis.

QED

3.3. Migrations on an Arbitrary Tree

We now consider the migration problem when the distance matrix has the property that it is the metric of a tree. That is, the access cost matrix $\{\delta_{ij}\}$ has that property that there exists a tree T, with lengths on its edges such that the distance between i and j in T is δ_{ij}. We also let m denote the ratio of the move cost to the access cost between any pair of nodes.

Algorithm **M-Tree** is a 3-competitive algorithm for this problem. Like algorithm **M**, this algorithm maintains a count c_i on each vertex i, and the vertices compete for the page by incrementing and decrementing the counts. These counts are initially zero, and always lie in the range $[0, 2m]$. Algorithm **M-Tree** also makes use of the underlying tree T.

> **Algorithm M-Tree:** Initialize all counts c_i to zero. Let s be the vertex with the page. The algorithm processes a request to vertex r as follows: If $r = s$ then the access is free, and the algorithm does nothing. Otherwise the access is accomplished, some counts are incremented, some are decremented, and finally the page may be moved.
>
> Let P be the path in T from s to r. The counts of the vertices of P (except s) are incremented. A *peripheral path* is a maximal path (one that can't be extended) that starts at s, continues with vertices that have non-zero counts (using only edges of T), and deviates from P as soon as it can. The counts of the vertices on a peripheral path but not on P are decremented.
>
> Finally, if any neighbor in T of the vertex with the page has a count of $2m$, then the page is moved to that neighbor, and the count on the new page location is set to zero. This process is repeated until no neighbor of the vertex with the page has a count of $2m$.

It will be convenient to think of the vertices as forming a rooted tree, with the location of the page s being the root. This defines the children and parent of each node. The counts maintained by algorithm **M-Tree** satisfy the following invariants:

- The counter of the vertex with the page is zero.
- The sum of the counters adjacent to s is at most $2m$.
- At a vertex v other than s, the sum of the counts of the children of v is at most c_v.

The proofs of these invariants are similar to those of Lemma 1, and are omitted. These invariants have several corollaries:

- All counter values are bounded by 0 and $2m$.
- When the server is about to be moved from s to s', $c_{s'} = 2m$, and the counts on all vertices in the tree on the s side of edge (s, s') are zero. (In other words, if the path from v to s' passes through s, then $c_v = 0$.)

We now prove that algorithm **M-Tree** is strongly competitive.

Theorem 4 *Let $\mathcal{A}$ be any on-line or off-line algorithm for the migration problem on a tree. For any sequence σ of requests algorithm* **M-Tree** *satisfies:*

$$\mathrm{C}_{\textbf{M-Tree}}(\sigma) \leq 3 \cdot \mathrm{C}_{\mathcal{A}}(\sigma)$$

under the assumption that $\mathcal{A}$ and **M-Tree** *start in the same state.*

Proof: We again partition what happens into a sequence of three types of events: algorithm **M-Tree** moves the page from a node to its neighbor, algorithm $\mathcal{A}$ moves the page from a node to its neighbor, and a request is satisfied by both algorithms. Again we shall give a non-negative (initially zero) potential function Φ such that the following inequality is satisfied for every type of event.

$$3 \cdot \Delta \mathrm{C}_{\mathcal{A}} - \Delta \mathrm{C}_{\textbf{M-Tree}} \geq \Delta\Phi$$

Summing this formula over all the events, and using the fact that the initial potential is no more than the final potential gives the theorem.

Let the location of **M-Tree**'s page be s and the location of $\mathcal{A}$'s page be t. Let the path from s to t be Q. Let δ_{ab} be the distance between a and b in the tree, and let $p(v)$ denote the parent of vertex v in the tree rooted at s. The potential function we shall use is:

$$\Phi = 3m\delta_{st} + \sum_{i \notin Q} 2c_i\delta_{ip(i)} - \sum_{i \in Q} c_i\delta_{ip(i)}$$

If the event is that $\mathcal{A}$ moves its page from t to t', then there are two cases, depending on whether the move is toward or away from s ($\Rightarrow s$ and $\Leftarrow s$ respectively). Since $\Delta C_{\mathcal{A}} = m\delta_{tt'}$ we need to show that $\Delta\Phi \leq 3m\delta_{tt'}$

$\Leftarrow s$: The potential undergoes two changes: the coefficient of $c_{t'}$ changes from $2\delta_{tt'}$ to $-\delta_{tt'}$, and $3m\delta_{t',t}$ is added. The net change is thus $(-3c_{t'}+3m)\delta_{t',t}$. (Here we have made use of the symmetry of δ.) Since $c_{t'}$ is non-negative, this quantity is bounded by $3m\delta_{t',t}$.

$\Rightarrow s$: This is the reverse of that above, and the change in potential is $(3c_t - 3m)\delta_{t't}$. Since c_t is bounded above by $2m$, this change is also bounded by $3m\delta_{t',t}$.

If the event is that **M-Tree** moves its page from s to s', then there are two cases depending on whether the move is toward or away from t. Since $\Delta C_{\textbf{M-Tree}} = m\delta_{ss'}$ we need to show that $\Delta\Phi \leq -m\delta_{ss'}$.

$\Leftarrow t$: Φ undergoes three changes: $c_{s'}$ is zeroed, its coefficient changes from 2 to -1, and $3m\delta_{ss'}$ is added. The contribution of $c_{s'}$ changes from $4m\delta_{ss'}$ to 0, so the net change in potential is $-m\delta_{ss'}$.

$\Rightarrow t$: Again Φ is changed in three ways: c_s is zeroed, its coefficient changes from -1 to 2, and $3m\delta_{ss'}$ is subtracted. The contribution of c_s changes from $-2m\delta_{ss'}$ to 0. The net change in potential is again $-m\delta_{ss'}$.

The most complicated part of the analysis deals with the costs of satisfying the requests. Let r be the vertex that is requested. Let T be tree rooted at vertex s, and let x be the lowest common ancestor of r and t in T. When the request is satisfied a cost is incurred by algorithm **M-Tree** and also $\mathcal{A}$. We shall associate these costs, as well as the change in potential that occurs as a result of the operation, to the vertices. The potential associated with a vertex i is either $2c_i\delta_{ip(i)}$ or $-c_i\delta_{i,p(i)}$ depending on whether i is on the path from s to t. A vertex i that is on the path from r to s (but is not s) gets a cost of $\delta_{i,p(i)}$ for algorithm **M-Tree**. A vertex i that is on the path from r to t gets a cost of $\delta_{i,p(i)}$ for $\mathcal{A}$. No other vertices incur cost. All the costs incurred by either algorithm, and all the potential changes are in this way is partitioned among the vertices.

Let PR be the path from r to x, let PT be the path from t to x, and let PS be the path from x to s. Furthermore, let P be the part of the peripheral path that is disjoint from the path from r to s. We shall examine the costs and potential changes incurred by each vertex, and show that it satisfies the inequality $3 \cdot \Delta C_{\mathcal{A}} - \Delta C_{\textbf{M-Tree}} \geq \Delta\Phi$. There are several cases to consider, depending on where our test vertex i is with respect to the PR, PS, PT, and P.

$i \in PR$: c_i is incremented, so $\Delta\Phi = 2\delta_{ip(i)}$. Furthermore $\Delta C_{\mathcal{A}} = \Delta C_{\textbf{M-Tree}} = \delta_{ip(i)}$. This verifies the inequality.

$i \in PS$: Again c_i is incremented, but this time the coefficient in the potential is -1, so $\Delta\Phi = -\delta_{ip(i)}$. Furthermore $\Delta C_{\mathcal{A}} = 0$ and $\Delta C_{\textbf{M-Tree}} = \delta_{ip(i)}$, so the inequality is verified.

$i \in PT$: In this case $\Delta C_{\mathcal{A}} = \delta_{ip(i)}$ and $\Delta C_{\text{M-Tree}} = 0$. The potential could increase by as much as $\delta_{ip(i)}$ if $i \in P$. (If $i \notin P$ then the potential will not change.) The inequality is certainly true in this case.

$i \notin PR \bigcup PS \bigcup PT$: The costs incurred by both algorithms are zero. The potential will decrease by $2\delta_{ip(i)}$ if $i \in P$, otherwise the potential will not change, and the inequality is verified.

It remains only to verify that the potential cannot be negative. Those vertices on that path from s to t contribute a negative amount to the potential. The most negative contribution they could make is $-2m\delta_{st}$. The initial term $3m\delta_{st}$ guarantees that the potential can never be negative.

QED

3.4. Migration on Uniform Trees

One disadvantage of algorithm **M-Tree** is that it must keep state for every node of the tree, even though the page can only be migrated to adjacent nodes. If the single edge distances in the tree are constant, we can collapse this state. Unfortunately, this collapsing of state disturbs the cost allocation of algorithm **M-Tree**, so instead of the strongly competitive factor of 3 we obtain a competitive factor of 4 for this algorithm. The algorithm still involves counters for every node of the tree, but the counters for nodes that are not adjacent to the copy of the page are always zero. For a tree whose nodes have at most k neighbors, at most k counters need to be maintained. As before we call the nodes adjacent to the page *boundary nodes*. We also assume without loss of generality that all single edge distances in the tree are 1. Our algorithm to solve the migration problem using boundary nodes is:

> **Algorithm M-UTree:** For each page P, initialize the counters c_i to zero. The algorithm processes a request from a node that does not have the page as follows: find the path to the page, and add its length to the counter for the boundary node on the path. Subtract as much of this path length as possible from the counters at the other boundary nodes without making any of them negative (i.e. the total of the decrements to the other counters does not exceed the path length, and is as large as possible without making any of the other counters negative). If the counter at the boundary node on the path is $\geq 2m$, migrate the page to the boundary node and zero its counter. If this counter was $> 2m$, set the counter at the new boundary node for the path to the original counter value less $2m$; if this new counter value is $\geq 2m$, the algorithm loops back to migrate the page to this new boundary node. If the request originated at the old boundary node, then the original counter was $2m - 1$ before the request and there is no excess value to be assigned.

Algorithm **M-UTree** maintains the invariant that the sum of the counters for any page at any node is bounded by 0 and $2m$.

The following theorem establishes that algorithm **M-UTree** is competitive with a competitive factor of 4:

Theorem 5 *For any sequence σ of requests for the tree page migration problem with uniform single edge access costs and any on-line or off-line algorithm $\mathcal{A}$*

$$\mathrm{C}_{\mathbf{M\text{-}UTree}}(\sigma) \leq 4 \cdot \mathrm{C}_{\mathcal{A}}(\sigma)$$

under the assumption that $\mathcal{A}$ and **M-UTree** *start in the same state with a single copy of each page.*

Proof: Assume without loss of generality that all single edge distances in the tree are 1 (i.e. if i and j are adjacent nodes, then $\delta_{ij} = 1$). Merge the actions taken by the two algorithms into a single sequence tagged to indicate which algorithms performed the actions. As before, we shall give a non-negative (initially zero) potential Φ such that the following inequality is satisfied by every event:

$$4 \cdot \Delta\mathrm{C}_{\mathcal{A}} - \Delta\mathrm{C}_{\mathbf{M\text{-}UTree}} \geq \Delta\Phi,$$

where Δ indicates the change in the quantity due to the event. Summing this formula over all events and using the fact that the initial potential is no maor than the final potential gives the theorem. It remains to verify the inequality for all events.

As in the previous proof, let s be the location of **M-UTree**'s page and t be the location of $\mathcal{A}$'s page. Let Q be the path from s to t. Let δ_{ab} be the distance between a and b in the tree and let $p(v)$ denote the parent of node v in the tree rooted at s. The potential function we shall use is:

$$\Phi = 3m\delta_{st} + \sum_{i \notin Q} 2c_i\delta_{ip(i)} - \sum_{i \in Q} c_i\delta_{ip(i)}$$

We now proceed to establish the desired inequality for all possible events:

If the event is that $\mathcal{A}$ moves its page from t to t' then there are four possible cases depending on the relative locations of the pages and whether the move is toward or away from s ($\Rightarrow s$ and $\Leftarrow s$ respectively). Since t and t' are adjacent, $\delta_{tt'} = 1$, so $\Delta\mathrm{C}_{\mathcal{A}} = m$ and we need to show that $\Delta\Phi \leq 4m$.

$\Leftarrow s,\ t = s$: The coefficient of $c_{t'}$ changes from 2 to -1. In addition $3m$ is added to Φ because the distance between the pages has increased by 1. Hence $\Delta\Phi = 3m - 3c_{t'} \leq 4m$ because the counters are non-negative.

$\Rightarrow s,\ t' = s$: The coefficient of c_t changes from -1 to 2. In addition, $3m$ is subtracted from Φ because the distance between the pages has decreased by 1. Hence $\Delta\Phi = 3c_t - 3m \leq 4m$ because $c_t \leq 2m$.

$\Rightarrow s,\ t, t' \neq s\ :\ \Delta\Phi = -3m \leq 4m.$

$\Leftarrow s,\ t, t' \neq s\ :\ \Delta\Phi = 3m \leq 4m.$

If the event is the **M-UTree** moves its page from s to s', then there are three cases. Let e be the new boundary node for the request that caused **M-UTree** to move its page; if there is no such node, then let c_e be zero. For the first two cases, $\Delta C_{\text{M-UTree}} = m$, so we must show $\Delta\Phi \leq -m$.

$\Leftarrow t$: $2m$ is subtracted from the sum of $c_{s'}$ and c_e. Since both have coefficients of 2 in Φ, this subtracts $4m$ from Φ. Since the distance between the severs increases by 1, $3m$ is added to Φ, so the net effect is $\Delta\Phi = -m$.

$\Rightarrow t,\ s' \neq t$: $2m$ is subtracted from the sum of $c_{s'}$ and c_e. Since both have coefficients of -1 in Φ, this adds $2m$ to Φ. Since the distance between the page decreases by 1, $3m$ is subtracted from Φ, so the net effect is $\Delta\Phi = -m$.

$\Rightarrow t,\ s' = t$: This page move must be analyzed in combination with the actions that satisfy the request that caused **M-UTree** to move its page. Let d be the length of the path for this request to s, $d \geq 1$. Then $\Delta C_{\mathcal{A}} = d - 1$ because it must perform an access from t which is closer than s. $\Delta C_{\text{M-UTree}} = m + d$ to account for both moving the page and the access from s. Let $c_t = 2m - x$ before the move where $1 \leq x \leq d$. Then after the move, $c_e = d - x$ since x of the d distance was needed to cause the move. c_t is zeroed as part of the move; since its coefficient in Φ is -1 before the move, this adds $2m - x$ to Φ. c_e's coefficient in Φ is 2, so it adds $2d - 2x$ to Φ. Finally $3m$ is subtracted from Φ because the distance between the pages decreases by 1. Substituting these into the desired inequality, we have

$$\begin{aligned} 4 \cdot \Delta C_{\mathcal{A}} - \Delta C_{\text{M-UTree}} &\geq \Delta\Phi \\ 4(d-1) - (m+d) &\geq (2m - x) + (2d - 2x) - 3m \\ 3d - m - 4 &\geq 2d - 3x - m \\ d + 3x &\geq 4 \end{aligned}$$

Since $d \geq 1$ and $x \geq 1$, the last inequality is always true, and the desired inequality is established for this case.

This leaves the actions that satisfy the requests. Let r be the node that originated the request. There are three cases depending on the relationship of that node to the pages positions s and t. Let e be the boundary node for algorithm **M-UTree** in all cases.

$s = t$: Both algorithms incur costs of $\delta_{rs} = \delta_{rt}$, so we must show $\Delta\Phi \leq 3\delta_{rt}$. δ_{rs} gets added to c_e. Since c_e's coefficient in Φ is 2, $\Delta\Phi \leq 2\delta_{rt} \leq 3\delta_{rt}$ because any subtractions from other counters decrease Φ.

$\delta_{rs} > \delta_{rt}$: $\Delta C_{\mathcal{A}} = \delta_{rt}$, $\Delta C_{\text{M-UTree}} = \delta_{rs} = \delta_{rt} + \delta_{ts}$, hence we must show $\Delta\Phi \leq 3\delta_{rt} - \delta_{ts}$. δ_{rs} gets added to c_e; since e is either t or between s and t, its coefficient in Φ is -1.

Hence $\Delta\Phi \leq \delta_{rs} = -\delta_{rt} - \delta_{ts} \leq 3\delta_{rt} - \delta_{ts}$ because any other subtractions decrease Φ.

$\delta_{rs} < \delta_{rt}$: $\Delta C_{\text{M-UTree}} = \delta_{rs}$, $\Delta C_{\mathcal{A}} = \delta_{rt} = \delta_{rs} + \delta_{st}$, hence we must show $\Delta\Phi \leq 3\delta_{rs} + 4\delta_{st}$. δ_{rs} is added to c_e; since its coefficient in Φ is 2, this adds $2\delta_{rs}$ to Φ. A further δ_{rs} is added to Φ if this entire amount is subtracted from $c_{e'}$ where e' is the boundary node on the path from s to t. Hence $\Delta\Phi \leq 3\delta_{rs} \leq 3\delta_{rs} + 4\delta_{st}$ since any other subtractions decrease Φ.

This completes the analysis of all possible actions, and therefore proves the theorem. QED

The disturbance to the cost allocation of **M-Tree** that produces the competitive factor of 4 for **M-UTree** occurs in the final case above; all of the other cases can be carried through for a competitive factor of 3. This disturbance is due precisely to the collapsing of the counters into the boundary node. For **M-Tree,** the decrement to c'_e would be spread out along the path from s to t (between the servers), and any addition to the potential would be matched by additional cost to $\mathcal{A}$, but for **M-UTree** it is possible to subtract more than this distance. This subtraction can not be offset against $\mathcal{A}$'s costs and hence requires a larger competitive factor.

3.5. Decrementation Variants

The policies for decrementing timers can be changed without affecting the competitive properties of the algorithms. The decrements used in the algorithms as stated are the minimum required to obtain the competitive properties; at most one counter is decremented after a counter increment. More aggressive decrementing can be performed in two ways without affecting the competitive properties:

1. Decrement more than one counter.
2. Decrement after free accesses (to the node with the server).

Both of these variants tend to discourage migration by subtracting more value from the counters than the original algorithms would.

Decrementing more than one counter will tend to avoid moving the page in response to a random access pattern by increasing the strength of accesses required to cause a migration in the presence of an overall random access pattern. At least one counter (or counters on one peripheral path for **M-Tree**) must be decremented if possible to preserve the competitive properties of these algorithms, but up to all of the eligible counters (i.e. non-zero and not incremented) may be decremented without destroying these properties. No counter may be decremented twice. For algorithm **M-UTree** this means that at most the distance of the access can be subtracted from each counter, and all counters must be non-negative after the decrements. For algorithm **M-Tree** the parent-child invariant (at a vertex v other than the location of the server, the sum of the counts of the children of v in a tree rooted at the server location is at most c_v) must be maintained by the decrements; an easy way to do this is to decrement at least one child count (if there is a non-zero one) when the parent is decremented. In the proofs, all of these decrements decrease the potential, except for decrements at the location of or on the path to $\mathcal{A}$'s server; in this case the potential increases are exactly matched by access costs that $\mathcal{A}$ must incur.

Decrementing after local (free) accesses will tend to leave the page situated at a node that is strongly accessing it; without this feature, a weak access pattern from some other node can cause the page to temporarily migrate away from a node that is accessing it strongly. It is not necessary to decrement any counters in response to a local access, but up to all of the non-zero counters may be decremented without destroying the competitive properties of the algorithms. As before, the decrements for algorithm **M-Tree** must maintain the parent-child invariant at all nodes in the tree. In the proofs, all of these decrements decrease the potential, except for decrements at the location of or on the path to $\mathcal{A}$'s server; in this case the potential increases are exactly matched by access costs that $\mathcal{A}$ must incur.

4. Look-Ahead One

All of the algorithms presented in this paper are look-ahead zero in that they may not look at the next access when making replication or migration decisions. An alternative model is look-ahead one, in which an algorithm may examine the next access but delay satisfying it until after one or more replication or migration actions have been performed. Look-ahead zero is a better match to the behavior of memory accesses in hardware, because it is unreasonable or impossible to delay satisfying a memory access while a page of data is copied between local memories. In contrast, some caching problems (e.g. General Snoopy Caching in [10]) are inherently look-ahead one because the algorithm can choose how to satisfy an access (fetch location remotely or fetch block from remote cache) after seeing it.

Our algorithms and results carry over to the look-ahead one model with minor changes. For replication, the algorithms remain strongly competitive with a competitive factor of two, and replications now occur in response to the first access after a node's counter reaches r. For migration, the lower bound result is weakened; we can only establish a lower bound on the competitive factor of $3(1 - 1/m)$. m is expected to be large, at least several thousand, so we don't consider this to be an important difference in practice. Modifying the algorithms to migrate on the first access to a node after the node's counter hits $2m$ (if it is not decremented in the interim) yields look-ahead one algorithms with the same competitive factors.

5. Applications of the Algorithms

The algorithms we have presented and analyzed are applicable to a significant collection of existing and proposed multiprocessors. Each node in the graphs used by the algorithms corresponds to a processor-memory cluster in a multiprocessor realization of that graph's interconnection topology. The primary hardware requirement for use of these techniques is that the hardware implement a shared memory model (i.e. support access forwarding). This excludes most current implementations of network shared memory on local area networks; in this case, access forwarding is not possible because page faults must be satisfied by data in a processor's local memory. This is also the case for most current hypercubes and related machines, although research has been conducted into similar machines that do support access forwarding [14].

A secondary requirement is that the reference counting information needed by our algorithms be available. There are several potential methods for doing this:

- A companion paper [2] proposes and analyzes a complete hardware implementation of our algorithms for **Complete**.
- Hardware reference counters (per cluster × per page) could be used in combination with a periodic software scan.
- Holliday [8] describes experiments that employed software-implemented usage counters based on periodic scans of page table reference bits.

In both cases involving experimental data, mixed results have been obtained for these [2] and similar [8] techniques. It is our opinion that software-implemented counters based on page table reference bits are sufficient for replication, but not for all cases of migration (in particular they are likely to fail to capture cases in which two clusters are actively using a page, but the usage in one cluster is more intensive than the other). We would recommend that multiprocessor architects and designers consider providing per-processor reference counters for some portion of the shared memory subsystem; this would allow implementation of our algorithms and make reference data available for other uses (e.g. hardware performance analysis).

A secondary issue that comes up in the area of reference counters is how references should be counted on machines with caches; in particular, should cache hits be considered. Removing cache hits from the reference counts removes a large amount of locality, but this corresponds exactly to the function of a cache; take advantage of locality to avoid loading the memory subsystem. Our algorithms apply to reference streams consisting entirely of cache misses and writebacks/writethroughs, so an implementation that counts only those references that reach memory is reasonable. Despite this there are two potential reasons to count cache references:

- At least one proposed research machine exhibits different cache behavior for remote and local pages (remote pages are uncacheable) [1]. An implementation on this hardware should count all local references that hit in cache because they would miss if the page were remote.
- Cache hit traffic may be a good predictor of cache miss traffic. This is an open question requiring further study.

It is certainly simpler from a hardware standpoint not to count cache hits; this allows the reference counters to be implemented in the memory proper, as opposed to the various caches.

There are many existing and proposed multiprocessors exhibiting the the **Complete** topology that are amenable to our algorithms and techniques. These machines include network-connected NUMA multiprocessors such as the the BBN Butterfly [3] and IBM RP3 [13], as well as bus-based machines such as the Encore Gigamax [15]. Numerous proposed machines such as the NYU Ultracomputer [6], and the directory-based cache machine (DASH) at Stanford [7] would also support our algorithms. In contrast, the **Tree** and **U-Tree** topologies are applicable to far fewer machines. The only existing machine that comes close is the experimental ACE multiprocessor developed by IBM's research division [5]. This machine is based on romp microprocessors with small local memories and a large global memory. The access ratio (local:global:remote) is an inverted triangle inequality in which the third side is longer than the sum of the other two. We have not thoroughly investigated extensions of our algorithms to this case or to the case of tree machines satisfying a triangle inequality (where it is cheaper to cross a node than to stop there and then move on); in both cases preliminary work has convinced us that the extensions are not straightforward. Tree-based machines using an architecture such as Fat-trees [11] would also be amenable to our techniques.

Our migration algorithms also extend to product topologies; the appropriate algorithm is run independently in each dimension of resulting topology. The most common examples of such topologies are hypercubes and meshes, which are products of linear trees; our algorithms for **Tree** and **U-Tree** apply to such machines. Scheurich and Dubois have independently discovered our migration algorithm for **U-Tree** and investigated it on a mesh machine; they were not aware of its competitive properties [14].

Using our migration techniques on rings and products involving rings (e.g. torii) is problematic due to cycling and pinning effects. Bidirectional effects exhibit the phenomenon of *pinning* in which accesses in both directions from the far side of the ring can pin a page in place and prevent it from moving towards the accesses. Unidirectional rings or unidirectional routing structures imposed on bidirectional rings avoid pinning, but exhibit the related phenomenon of *cycling* in which a static access pattern distributed over the ring can cause a page to cycle around the ring when it should stay put. These effects would cause the size of the ring to enter into the competitive factor for the straightforward extensions of our algorithms to these topologies; a more sophisticated approach is needed.

6. Further Work

The primary problem of interest from a theoretical standpoint is the migration problem (1-server with excursions). This paper reports the first work to be done on that problem, so competitive algorithms for migration on other topologies is an open area for research. The authors have used the techniques developed in [12] to investigate some small graphs (other than those considered in this paper) whose distance metrics satisfy the triangle inquality. Our results indicate that 3-competitive algorithms exist for the small examples investigated. Based on our results and experience, we believe the following conjecture to be true:

Conjecture: *There exists a 3-competitive algorithm for the migration problem for any topology having a symmetric distance matrix that satisfies the triangle inequality.*

Another direction for extensions of this work is to consider randomized algorithms for the migration problems. For the randomized model of competitiveness, the on-line algorithm is allowed to make use of random choices. The cost incurred by the randomized algorithm on a sequence of requests is defined to be the average of its costs over all of the possible series of random choices. Competitiveness is defined as before, but it uses this modified definition of cost. For a number of different problems it has been shown that the competitive factor can be reduced by the use of randomness [4,9].

7. Conclusion

This paper has presented and analyzed new strongly competitive algorithms for replication and migration problems that arise in the management of distributed shared memory for multiprocessor systems. These algorithms are applicable to many existing and proposed multiprocessor architectures. The proofs of the competitive properties of the algorithms have also served to establish new results in the area of competitive algorithm analysis for server problems. We have also briefly highlighted some of the issues involved in actually applying these algorithms to real systems.

References

[1] Roberto Bisiani, Andreas Nowatzyk, and Mosur Ravishankar. *Coherent Shared Memory on a Message Passing Machine.* Technical Report CMU-CS-88-204, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1988.

[2] David Black, Anoop Gupta, and Wolf-Dietrich Weber. Competitive Management of Distributed Shared Memory. In *Proceedings, Spring Compcon '89*, pages 184–190, IEEE Computer Society, San Francisco, CA, February 1989.

[3] W. Crowther, J. Goodhue, E. Starr, R. Thomas, W. Milliken, and T. Blackadar. Performance Measurements on a 128-node Butterfly Parallel Processor. In *Proceedings of the International Conference on Parallel Processing*, pages 531–540, IEEE Computer Society, 1985.

[4] Amos Fiat, Richard Karp, Michael Luby, Lyle McGeoch, Daniel Sleator, and Neal Young. *Competitive Paging Algorithms.* Technical Report CMU-CS-86-164, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1986.

[5] Armando Garcia, David Foster, and Richard Freitas. *The Advanced Computing Environment Multiprocessor Workstation.* Research Report RC14491, IBM T. J. Watson Research Center, Hawthorne, NY, 1988.

[6] Alan Gottlieb. The NYU Ultracomputer - Designing a MIMD Shared-Memory Computer. *IEEE Transactions on Computers*, C-32(2):175–189, February 1983.

[7] John Hennessy. A Scalable Shared Memory Architecture. Talk presented at Carnegie Mellon University, April 1989.

[8] Mark Holliday. Reference History, PageSize, and Migration Daemons in Local/Remote Architectures. In *ASPLOS-III Proceedings*, ACM/IEEE Computer Society, Boston, MA, April 1989.

[9] Anna Karlin, Mark Manasse, Lyle McGeoch, and Susan Owicki. Competitive Randomized Algorithms for Non-Uniform Problems. In *Proceedings, Symposium on Discrete Algorithms, 1990*, ACM-SIAM, San Francisco, CA, January 1990.

[10] Anna Karlin, Mark Manasse, Larry Rudolph, and Daniel Sleator. Competitive Snoopy Caching. *Algorithmica*, 3(1):79–119, 1988.

[11] Charles Leiserson. Fat Trees: Universal Networks for Hardware-Efficient Supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, October 1985.

[12] Mark Manasse, Lyle McGeoch, and Daniel Sleator. Competitive Algorithms for On-line Problems. In *Proceedings of the 20th ACM Symposium on the Theory of Computing*, pages 322–333, ACM SIGACT, May 1988.

[13] G. Pfister, et. al. The IBM Research Parallel Processor Prototype: Introduction and Architecture. In *Proceedings of the International Conference on Parallel Processing*, pages 764–771, IEEE Computer Society, 1985.

[14] Christoph Scheurich and Michel Dubois. Dynamic Page Migration in Multiprocessors with Distributed Global Memory. *IEEE Transactions on Computers*, 38(8):1154–1163, August 1989.

[15] Andrew Wilson Jr. Hierarchical Cache/Bus Architecture for Shared Memory Multiprocessors. In *Conference Proceedings, 14th International Symposium on Computer Architecture*, pages 244–252, ACM SIGARCH/IEEE Computer Society, Pittsburgh, PA, June 1987.