DISTRIBUTED ALGORITHMS FOR DYNAMIC REPLICATION OF DATA

Preliminary Version

Ouri Wolfson[1]
Department of Electrical Engineering and Computer Science
University of Illinois
Chicago, IL 60680
wolfson@dbis.eecs.uic.edu

Sushil Jajodia[2]
Information and Software Systems Engr. Department
George Mason University
Fairfax, VA 22030-4444

ABSTRACT

We present two distributed algorithms for dynamic replication of a data-item in communication networks. The algorithms are adaptive in the sense that they change the replication scheme of the item (i.e. the set of processors at which the data-item is replicated), as the read-write pattern of the processors in the network changes. Each algorithm continuously moves the replication scheme towards an optimal one, where optimality is defined with respect to different objective functions. One algorithm optimizes the communication cost objective function, and the other optimizes the communication time. We also provide a lower bound on the performance of any dynamic replication algorithm.

1. Introduction

The data replication scheme of a distributed database determines how many replicas of each data item[3] are created, and to which processors these replicas are allocated. This scheme critically affects the performance of a distributed system, since reading a data-item locally, or close by, is faster than reading it from a remote processor. Therefore in a read-intensive network a widely distributed replication is mandated. On the other hand, an update of a data-item is usually written to all the replicas, and therefore in a write-intensive network a narrowly distributed replication is mandated. In other words, the optimal replication scheme depends on the read-write pattern for each item.

Currently, the data replication scheme is normally established in a static fashion, when the distributed-database is designed, and it remains fixed until the designer manually intervenes to change the number of replicas or their location. If the read-write patterns are fixed and are known a priori, this is a reasonable solution. However, if the read-write patterns change dynamically, in unpredictable ways, a static replication scheme may lead to severe performance problems.

In this paper we propose two practical algorithms, COST-ADAPTIVE REPLICATION (CAR) and TIME-ADAPTIVE-REPLICATION (TAR). They change the replication scheme of a data-item (i.e. the set of processors, each of which stores a replica of the data-item) dynamically as the read-write pattern of the item changes in the network. We assume that the changes in the read-write pattern are not known a priori.

The algorithms are **distributed** as opposed to centralized. In a centralized algorithm, each processor periodically transmits the relevant information (usually statistics) to some predetermined processor, x. In turn, x computes the objective function and orders the change of replication scheme. In contrast, in a distributed algorithm, each processor makes decisions to locally change the replication scheme, and it does so based on statistics collected locally. An example of a local change to the allocation scheme is the following: a processor, x, relinquishes a copy of the data-item (by indicating to x's neighbors that writes of the item should not be propagated to x). This change may result from a comparison of the number of reads to the number of writes, at x (i.e. locally collected statistics).

Distributed algorithms have two advantages over centralized ones. First, they respond to changes in the read-write pattern in a more timely

[1] This research was supported in part by NSF grant IRI-90-03341.

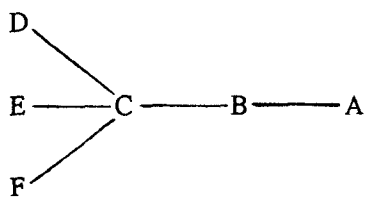
[2] This research was supported in part by grant AFOSR 90-0135.

[3] A data item may be an object, a file, a relation, etc

11th Principles of Database Systems/6/92/San Diego, CA
© 1992 ACM 0-89791-520-8/92/0006/0149...$1.50

manner, since they avoid the delay involved in the collection of statistics, computation, and decision broadcast. Second, their overhead is lower.

Although in our algorithms the decision-making and schema-changing are *local*, each algorithm continuously moves the replication scheme towards an optimal one, where optimality is defined with respect to a *global* objective function. The CAR algorithm optimizes the communication cost objective function. The communication cost of a replication scheme is the average number of inter-processor messages required for a read or a write of the data item. The TAR algorithm optimizes the communication time objective function. The communication time of a replication scheme is the average communication time of a read or write of the data item (the communication time of an operation is its longest message path). The algorithms work in the read-one-write-all context (see [BHG, CP, OV]). Specifically, a read of the data-item is performed from the closest replica in the network. A write is to all the replicas, and it is propagated along the edges of a subtree that contains the writer and the processors of the replication scheme. For example, consider the communication network T of the figure below, and suppose that the replication scheme consists of processors C, D and E.



When processor A writes the item, A sends the item to B, then B sends it to C, and then C sends it to D and E simultaneously. Overall, the communication cost of this write is four, namely the total number of inter-processor messages required. The communication time of the write is three, namely the longest message path (A — B — C — D or A — B — C — E).

The algorithms are **optimal** in the following sense. Assume that the read-write pattern of each processor is generally regular. For example, during the first two hours processor A executes three reads and one write per second, processor B executes five reads and two writes per second, etc.; during the next four hour period processor A executes one read and one write per second, processor B executes two reads and two writes, and so on. Then, we show that the CAR and TAR algorithms will converge to the optimal replication scheme for the global read-write pattern during the first two hours, then it will converge to the optimal replication scheme for the global read-write pattern during the next four hours, etc.

The proposed algorithms are also **integrated.** In such an algorithm, redistribution of the replicas is integrated into the processing of reads and writes, instead of executing independently of these operations. So, for example, a processor x creates a replica of a data-item at some neighbor, y, in response to y's request to read the data-item from x. x creates the replica by piggybacking, on the message to y containing the data-item, an indication that y should keep a replica (and that x will propagate writes to y).

In order to put the proposed algorithms in perspective, we examine and devise the lower bound on the communication cost and the communication time objective functions. The lower bound is the cost (time) of the algorithm that has complete knowledge of all the future read-write requests, and their order. A similar approach was taken by Belady in studying page replacement algorithms ([B]).

Before outlining the rest of the paper, we will make a few remarks concerning relevant work. Generally, there are two main purposes for data-replication: performance and reliability. In this paper we address the performance issue. Research works that concentrate on reliability usually devise more sophisticated policies than the read-one-write-all we assume here, in order to cope with failures (see for example [AE2, DGS, E, GB, H]). However, in the absence of failures, reliability-oriented works assume a fixed replication scheme. Performance-oriented works on replicated data consider the static problem of replication, namely establishing a priori a replication scheme that will optimize performance, but will remain fixed at runtime. This is called the file-allocation problem, and it has been studied extensively in the literature (see [DF] for a survey). Another approach to improve the performance in a replicated distributed database is to relax the serializability requirement. Works on quasi-copies ([ABG1, ABG2, BG]), lazy replication ([LLS]), and bounded ignorance ([KB]) fall in this category. These works also assume a static replication scheme

To the best of our knowledge this paper is the first to address the problem of dynamic (vs. static) data replication, although the need for dynamic replication was already pointed out in [GS]. In the theoretical computer science community there has been work on online algorithms (e.g. [BLS]), particularly for paging (e.g. [BS, CL, ST]), searching (e.g. [ST]) and caching (e.g. [KMRS]). These works are similar in spirit to the algorithms we discuss here, however, the models in such works is inappropriate for managing replication in distributed databases. For example, the notion of "distance" that arises in replicated data management does not arise in caching, and the replacement problem (i.e. which page to replace), that is important in paging, does not arise in replicated data management.

The rest of the paper is organized as follows. In section 2 we present, demonstrate and discuss the COST-ADAPTIVE-REPLICATION algorithm. In section 3 we prove that when the the replication scheme managed by the CAR algorithm stabilizes, then the scheme is optimal

for the communication-cost objective function. In section 4 we present the TIME-ADAPTIVE-REPLICATION algorithm and prove for it the counterpart of the result in section 3 (concerning the communication-time objective function). In section 5 we provide the lower bound and in section 6 we summarize the results. In appendix A we show that the CAR algorithm stabilizes whenever the read-write pattern of each processor becomes regular, and in appendix B we prove theorem 2.

2. The COST-ADAPTIVE-REPLICATION Algorithm

The CAR algorithm works for a tree network. Metaphorically, the replication scheme of the algorithm forms a variable-size amoeba that stays connected at all times, and constantly moves towards the "center of read-write activity". The replication scheme expands as the read activity increases, and it contracts as the write activity increases. When at each "border" processor (i.e. processor of the circumference of the amoeba) the number of reads equals the number of writes, the replication scheme remains fixed. Then this scheme is optimal for the read-write pattern in the network.

Specifically, the algorithm works as follows. The initial replication scheme consists of the whole set of processors. At any time, the processors of the replication scheme, R, induce a connected subgraph of the network. For example, the CAR algorithm will never replicate the item only at processors A and C of the network T in the introduction. Each processor i that is a *R-neighbor*, i.e. i belongs to R but it has a neighbor that does not belong to R, performs the following test for each read request from a processor j that is not in R.

(Expansion-Test) If there are two read requests made by j[4], between which i receives no write request made by i or by a neighbor different than j, then i tells j to join R.

Practically, j joins R simply by saving a copy of the item sent by i as a result of the second consecutive read request. Except for i and j, no other processor is informed of the expansion.

Suppose that R is not a singleton set. Then each processor j that is an *R-fringe* node, i.e. a leaf of the subgraph of the network induced by R, performs the following test for each write request from the single neighbor i that is in R.

(Contraction-Test) If there are two write requests made by i, between which j receives no read request made by j or by a neighbor of j, then j exits R, i.e. ceases to keep a copy.

j exits R by telling i not to send any more write requests to j; any further read requests arriving at j are passed along to i. Except for i and j, no other processor is informed of the contraction.

Finally, the following test is executed by a processor that constitutes the whole replication scheme, R, of the algorithm. Namely, a processor i that is in R but none of its neighbors is in R, performs the following test for each read or write operation from some neighbor n.

(Switch-Test) If i executes two operations o_1 and o_2, requested by n, such that o_1 or o_2 (or both) is a write and between them i executes no operation requested by i or by a neighbor other than n, then the single copy of the item is transferred from i to n (i.e. simultaneously, i exits R and n enters it).

To summarize the algorithm, each R-neighbor performs the Expansion-Test on a read request; an R-fringe node that has some neighbor in R performs the Contraction-Test on a write request, and if it does not have a neighbor in R then it performs the Switch-Test on a read or a write request. Note that a node may be both an R-fringe node and a R-neighbor.

Let us demonstrate the algorithm using the following example.

Example 1: Consider the network T of the introduction. The initial replication scheme, R, consists of the whole set of processors in the network. Suppose that the first request is a write that originates at D. After it is received by all the processors, the second request is initiated. The second request is a write that originates at E. After the execution of the second write by all the processors, the R-fringe nodes F and A exit the replication scheme, R. For example, A exits since the first two requests were presented to A by B. Then the third request, a write that originates at B, is initiated (after its execution R = {B, C}; E and D exit since each has executed two write requests from C, without an intervening read). The fourth request is a read that originates at A (after its execution R remains {B, C}). The fifth request is a write that originates at A (after its execution R = {B}). The sixth request is a read that originates at A (R = {A, B}). The seventh request is a read that originates at D (R = {A, B}). The eighth request is a read that originates at F (R = {A, B, C}). []

For the rest of this section we will discuss the CAR algorithm. Note that the algorithm requires that each node of R knows whether it is a R-neighbor, or an R-fringe node, or a unique node of R. Knowing this requires only that the node knows the identity of its neighbors, and "remembers" for each neighbor whether or not it is in the replication scheme. A processor that does not belong to the replication scheme does

[4] j makes read and write requests, each of which originates either in j or in some other node, k, such that the shortest path from k to R goes through j. For example, consider the network T. If R = {C}, then B makes read-write requests that originate in B or A.

not participate in the algorithm; nor does an internal node of the replication scheme, i.e. a node that is not an R-fringe, and that does not have a neighbor outside R.

What does a node need to know in order to execute reads and writes? A node j of the current replication scheme, R, satisfies a read request locally, and transmits each write request to the neighbors that are in the replication scheme (each of which in turn, propagates the write to its neighbors that are in R, except j). Therefore, j has to know the identity of its neighbors, and to remember for each neighbor whether or not it is in the replication scheme (same as the information needed to execute the CAR algorithm). Interestingly, a node that does not belong to the replication scheme does not have to "search" for the replication scheme in order to execute reads and writes. A node j that is not in the replication scheme R, must remember the node i to which j sent the last announcement that j exits R. i indicates the "direction" of R. Each read or write of j must be sent to i, which in turn, if is not in R any longer, routes the request in the "direction" of R. Therefore, for executing the CAR algorithm and for executing reads and writes, knowledge of the whole network topology is not necessary; nor is knowledge of the whole replication scheme necessary.

Practically, each node, j, has a directory-record for each data-item. The record indicates whether or not j is in the replication scheme, R; if it is, then the record also indicates which of j's neighbors are in R, and if it is not, then it indicates the direction of R. Before accessing a data-item, j accesses its directory record. This access is part of the transaction that accesses the data-item, and consequently a concurrency control mechanism that ensures serializability of transactions in a static replication environment will also do so in this dynamic environment. The reason is that a transaction that (indirectly) changes the replication scheme (e.g. it expands it) can complete only after updating the directory record. This update conflicts with any read of the directory record by a transaction that reads or writes the data-item.

The next comment concerns the exit of a node j from the replication scheme, as a result of the Contraction-Test. j does not leave R unconditionally, simply by announcing j's exit to its neighbor i in R. The reason is that i and j may be the only nodes of the current replication scheme, and j and i may both announce their exit to each other, leaving an empty replication scheme. Therefore, if the Contraction-Test succeeds then j requests permission from i to exit R, but j keeps a copy of the data-item until it receives the next message from i. If the next message from i is i's request to leave R then only one (say the one with the smaller processor-identification-number) leaves R.

Finally let us mention that, in contrast to [AE1], our model ignores the cost of storing a replica. Therefore, in practice, our algorithm is optimal only in an environment in which storage costs are negligible.

3. Analysis of COST-ADAPTIVE-REPLICATION

In this section we formally define the model and analyze the CAR algorithm. A *network* is an undirected tree, $T=(V,E)$. V represents a set of processors, and an edge in the network between two processors represents a bidirectional communication link between them. The *replication scheme* is a nonempty subset of V. For a given network and replication scheme, the *read cost* of a processor i, denoted r_i, is the length (in edges) of the shortest path in the network between i and a processor of the replication scheme. It represents the number of messages required for the data-item transfer. Obviously, if i is in the replication scheme, then the read cost is zero. Let R be a replication scheme, and assume that processor $i \in V$ writes the data-item. The *write cost* for i, denoted w_i, is the number of edges in the smallest subtree of T that contains $\{i \cup R\}$.

A *schedule* is a sequence $o_1^{j_1}, o_2^{j_2}, \ldots o_n^{j_n}$. Each o_i is a read or write operation, and each j_i is a processor of V, at which the operation o_i originated. Intuitively, each operation represents the initiation of the request, as well as its execution. In other words, we assume in particular that each write request is executed by all the processors of the replication scheme before the next request is issued. It is important to emphasize that we make this assumption for the purpose of analyzing the CAR algorithm, but the correctness of the algorithm does not depend on the completion of each operation before issuing the next one. Given a schedule, $o_1^{j_1}, o_2^{j_2}, \ldots o_n^{j_n}$, a *subschedule*, S, is a subsequence $o_i^{j_i}, o_{i+1}^{j_{i+1}}, \ldots o_{i+k}^{j_{i+k}}$. The set of pairs $A = \{ (\#R_i, \#W_i) \mid i$ is a processor in the network, and $\#R_i$ and $\#W_i$ are nonnegative integers$\}$, is called a *read-write-pattern*. Suppose that S is a subschedule, and in S each processor, i, of the network performs $\#R_i$ reads, and $\#W_i$ writes. The set of pairs $A = \{ (\#R_i, \#W_i) \mid i$ is a processor in the network$\}$, is called the *read-write-pattern of* S. Given a replication scheme, R, and a read-write pattern, A, the *replication scheme cost* for A, denoted $cost(R,A)$, is defined as $\sum_{i \in V} \#W_i \cdot w_i + \sum_{i \in V} \#R_i \cdot r_i$. Intuitively, $cost(R,A)$ represents the total number of messages sent during the subschedule, assuming that R is the replication scheme. A message is the transmission of the data-item over one communication link (edge). A replication scheme is *cost-optimal* for a read-write pattern, A, if it has the minimum (among all replication schemes) cost for A. Obviously, cost optimality of the replication scheme implies that the average cost of an operation in A is minimum.

Next we define a dynamic replication algorithm. A *configured–schedule*, $o_1^{j_1}(R_1), o_2^{j_2}(R_2), \ldots o_n^{j_n}(R_n)$ is a schedule in which each operation is mapped to a replication scheme. Intuitively, it is the replication scheme that exists when the operation is initiated. We assume that any recomputation of the replication scheme is executed before the next operation is initiated. For the above configured schedule, the replication scheme R_i is *associated* with the operation $o_i^{j_i}$. A *dynamic replication algorithm* is a function that maps each schedule to a configured schedule.

Let S' be a schedule and let DRA be a dynamic replication algorithm. Suppose that S' is mapped by DRA to the configured schedule S''. Let S be a subschedule of S''. Suppose that the operations of S and the operation immediately succeeding the last one in S, are all associated with the same replication scheme R. Intuitively this means that S starts and ends with the same replication scheme. Then we say that DRA is *stable* on S, with *stability scheme* R.

Now consider the CAR algorithm. It moves towards the center of read-write activity in the following sense. Suppose that at some point in time, t, all the processors become quiescent (i.e. stop initiating operations), except for one, i; and, suppose that i issues only reads. Then it is clear intuitively that the CAR algorithm will stabilize, and that the stability scheme will include i (i.e. it will be optimal for any subschedule consisting only of reads from i). Specifically, if at time t processor i is in the replication scheme, then as long as it issues read requests and all the other processors are quiescent, the replication scheme will not change. If processor i is not in the replication scheme, R, then R will expand towards i until it reaches i, and from then on the CAR algorithm will become stable. Each expansion step will take two reads, and therefore, the convergence to the optimal replication scheme will occur after a number of reads that is bounded by twice the number of processors in the network.

Now suppose that from point in time t on, processor i issues only writes. Then the CAR algorithm will stabilize, and the stability scheme will be optimal for any subschedule consisting only of writes from i (i.e. the stability scheme will be the singleton set, $\{i\}$). Specifically, if at time t processor i is in the replication scheme, then as long as it issues write requests and all the other processors are quiescent, the replication scheme will contract until it consists of the single processor, i. If processor i is not in the replication scheme, R, then denote by j the processor of R that is closest to i. R will contract until it consists of the singleton set $\{j\}$, and then it will switch until it consists of the singleton set $\{i\}$. In both cases, convergence to the optimal replication scheme will occur after a number of writes that is bounded by twice the number of processors in the network.

Therefore, for subschedules consisting of a single operation, the CAR algorithm stabilizes, with a stability scheme that is optimal for the operation. However, the CAR algorithm will stabilize not only when all processors except one quiesce, but whenever the read-write pattern of each processor becomes regular (e.g. processor 1 executes two reads and one write every time-unit, processor 2 executes three reads and two writes every time-unit, etc.) This is shown in appendix A. The following theorem states that when the CAR algorithm stabilizes, the stability scheme is (almost) optimal for the read-write pattern in the network.

Theorem 2: Let S' be a schedule that is mapped by the algorithm COST-ADAPTIVE-REPLICATION to a configured-schedule, S''. Let c be the number of processors in the tree network, and let S be a subschedule of S'', in which each processor of the network performs at least one operation. Let A be the read-write pattern in S. If the CAR algorithm is stable on S with stability scheme R, then $cost(R,A)$ is higher than the cost of an optimal replication scheme for A by at most c.

Proof: See APPENDIX B.

Theorem 2 indicates that the difference (not ratio) between the cost of the stability scheme and the optimal cost is at most the number of nodes in the network. This holds regardless of the length of the schedule S, and the cost of its read-write pattern. Obviously, the higher this cost, the more insignificant the difference becomes.

Note also that the bound in theorem 2 is tight. For example, consider a star network, and the following schedule: $S = w, Q, w, Q, w, Q, \ldots\ldots, Q, w$. In this schedule w denotes a write of the hub of the star, and Q represents a sequence of reads, one from each leaf. Starting with the full replication scheme, that consists of the whole set of processors, the CAR algorithm is stable on the schedule S. However, the optimal replication scheme is the singleton consisting of the hub alone. Furthermore, the difference between the costs of the two replication schemes (full and singleton) is equal to the number of leaves. However, for an arbitrarily long schedule, the ratio between the costs is arbitrarily close to one.

4. The TIME-ADAPTIVE-REPLICATION Algorithm

Our TAR algorithm works for a star network, i.e. a network consisting of a hub-node, h, and multiple leaves, each of which is connected by an edge to the hub. Star is an important topology from the practical point of view, since it models a set of workstations connected to a mainframe. Notice that the time of a read or a write operation in a star network is either 0, 1, or 2.

For simplicity we shall assume that there are at least two leaves in the star. The initial replication scheme consists of the whole set of processors. Subsequently, the replication scheme,

R, is one of four possible configurations. R may consist of the hub alone, or it may consist of the hub and one leaf, or it may consist of one leaf alone, or it may consist of all the processors in the network. Let us call these four configurations *hub*, *two*, *leaf*, and *all*, respectively. Notice that the TAR algorithm will never replicate the item at more than one, but not all the leaves of the network (in contrast to the CAR algorithm, that may produce such a replication scheme). Intuitively, the reason for this is that if the item is replicated at two or more leaves, then the time for each write from a leaf is two, and the time for a write from the hub is one. But if the item is replicated at all the leaves, then the write times are identical, and the read times are not higher (and possibly lower).

Now let us describe the algorithm TIME-ADAPTIVE-REPLICATION. At any time during the execution of the algorithm, each node knows the current configuration. The algorithm executed by the leaf of the replication scheme in the *two*, and *leaf* configurations is identical to the procedure of a processor in the CAR algorithm. Namely, in the *leaf* configuration, the leaf that stores the single replica executes the expansion and switch tests of the CAR algorithm; in the *two* configuration the leaf in the replication scheme executes the contraction test (the expansion test is skipped since the leaf does not have a neighbor outside the replication scheme).

In the *all* configuration each leaf simply counts the number of reads between every pair of consecutive writes. The tests executed by the hub, h, for a write operation in the all configuration are All-contraction-test1 and All-contraction-test2, in this order.

(All-contraction-test1) If h executes two consecutive writes w_1 and w_2, both of which are requested by some leaf, i, such that between w_1 and w_2 no leaf (except possibly i) requests a read, then h tells all the leaves, except i, to exit from the replication scheme.

How does h know the number of reads at the leaves? The answer is that each leaf, i, in response to a write propagated to it by h, sends to h the number of reads i executed since the previous write. If the write is initiated by a leaf, j, then j piggybacks this information on the write request.

(All-contraction-test2) If h executes two consecutive writes w_1 and w_2, such that between w_1 and w_2 no leaf requests a read, then h tells all the leaves to exit from the replication scheme.

Suppose the replication scheme, R, consists of h alone. Then h executes Hub-expansion-test1, Hub-expansion-test2, and Hub-switch-test.

(Hub-expansion-test1) If h executes two consecutive reads r_1 and r_2, requested by different leaves, such that between r_1 and r_2 no write is requested, then h tells all the leaves to join R (i.e., in response to r_2, h sends the item to all the leaves).

(Hub-expansion-test2) If h executes two consecutive reads, r_1 and r_2, both of which are requested by some leaf, i, such that between r_1 and r_2 no processor (except possibly i) requests a write, then h tells i to join R.

(Hub-switch-test) Suppose h executes two consecutive operations, o_1 and o_2, at least one of which is a write and both of which are requested by the same leaf, i, such that between them h executes no operation requested by h or by a neighbor other than i. Then the single copy of the item is transferred from h to i (i.e. simultaneously, h exits R and i enters it).

Suppose that the replication scheme, R, consists of h and one leaf, i. Then h executes Two-expansion-test and Two-contraction-test.

(Two-expansion-test) Suppose h executes two consecutive reads, r_1 and r_2, both of which are requested by leaves, such that between them no write is requested by i, then h tells all the leaves to join R (i.e., in response to r_2, h sends the item to all the leaves).

(Two-contraction-test) Suppose h executes two consecutive writes, w_1 and w_2, both of which are requested by i, such that between them no processor (including h) requests a read from h, then h exits from R.

For a given star network and a given replication scheme, the *read time* of a processor i, denoted r_i, is the length (in edges) of the shortest path in the network between i and a processor of the replication scheme (same as the read-cost). The *write time* for i, denoted w_i, is the number of edges in the longest path from i to a member of the replication scheme.

Given a replication scheme, R, and a read-write pattern, $A = \{ (\#R_i, \#W_i) \}$ the *replication scheme time* for A, denoted $time(R,A)$, is defined as $\sum_{i \in V} \#W_i \cdot w_i + \sum_{i \in V} \#R_i \cdot r_i$. Intuitively, $time(R,A)$ represents the total communication time of operations in A, assuming that R is the replication scheme. A replication scheme is *time-optimal* for a read-write pattern, A, if it has the minimum (among all replication schemes) time for A.

Theorem 3: Let S' be a schedule that is mapped by the algorithm TIME-ADAPTIVE-REPLICATION to a configured-schedule, S''. Let S be a subschedule of S'', in which each processor of the network performs at least one operation. Let A be the read-write pattern in S. If TIME-ADAPTIVE-REPLICATION is stable on S with stability scheme R, then $time(R,A)$ is higher than the time of an optimal replication scheme for A by at most four.

5. A Lower Bound for Dynamic Replication Algorithms

The ADAPTIVE replication algorithms are "online" in the the sense that after each request the new replication scheme has to be determined without knowledge of what is the next request. Furthermore, the new replication scheme is determined in a distributed fashion. In this section we present the optimal centralized offline algorithm, called LOWER-BOUND (LB). Suppose that the input consists of a schedule. The algorithm LB has to optimally configure it, i.e., it has to produce a configured schedule with minimum cost (see sec.3 for the definition of a configured schedule). The *cost* of a configured schedule is the cost of the read-write operations, plus the minimum cost of moving the item from one replication scheme to the next, if the latter is associated with a read (for a write, the item does not have to move to the new replication scheme, since it will be overwritten by the write operation). So, for example, consider the configured schedule: $r_1^4(\{1,2,3\}), r_2^5(\{7,8,9\})$. Its cost is:

[the cost of a read by 4 from the scheme {1,2,3}]
+
[the minimum cost of moving from {1,2,3} to {7,8,9}]
+
[the cost of a read by 5 from the scheme {7,8,9}]

For the schedule: $r_1^4(\{1,2,3\}), w_2^4(\{7,8,9\})$, the cost is:

[the cost of a read by 4 from the scheme {1,2,3}]
+
[the cost of a write by 4 to the scheme {7,8,9}].

Suppose that the network is an arbitrary tree. Given a schedule, its *cost-optimal* configured schedule is the one with minimum cost. Intuitively, in a cost-optimal configured schedule each write "guesses" the processors that will read from it. Precisely, the algorithm LB is as follows. It associates the replication scheme that consists of the whole set of processors with all the reads up to the first write. Additionally, LB associates with each write, w_i^j, the replication scheme consisting of: $\{j\} \cup$ {the processors that read the data-item between w_i^j and the first write that succeeds w_i^j}. For example, consider the schedule: $w_1^2, r_2^4, r_3^5, w_4^5, r_5^1$. The configured schedule devised by LB is: w_1^2,($\{2,4,5\}$), $r_2^4(\{2,4,5\})$, $r_3^5(\{2,4,5\})$, $w_4^5(\{1,5\})$, $r_5^1(\{1,5\})$. The cost of the configured schedule is:

[The cost of the minimum subtree
that includes the nodes 2, 4, and 5]
+
[The cost of the minimum subtree
that includes the nodes 1 and 5]

Theorem 4: Given a schedule, the algorithm LB produces the cost-optimal configured schedule.

The time of a configured schedule can be defined in a similar fashion.

Theorem 5: Given a schedule, the algorithm LB produces the time-optimal configured schedule.

There are subschedules for which the cost (time) of the CAR (TAR) algorithm is positive, whereas the cost of (time) of LB is zero. One such schedule consists of writes only, originating from different nodes in the network. The cost (time) of the optimal configured schedule is zero, since no processor issues a read, and therefore the write does not have to be transmitted. However, an algorithm that is not aware of future requests must transmit the write, therefore the cost (time) of the schedule produced by the CAR (TAR) algorithm is positive.

We conjecture that there does not exist an algorithm, A, such that: there are constants, C_1 and C_2, such that cost(A) < $C_1 \cdot$ optimum-cost + C_2, for every subschedule. However, even if such an algorithm exists one will probably choose the TAR and CAR algorithms, if the read-write pattern in the network is regular for long periods of time.

6. Conclusion

In this paper we proposed and analyzed dynamic replication algorithms. One is the COST-ADAPTIVE-REPLICATION algorithm, that works in a tree network, and converges to a cost-optimal replication scheme when the read-write pattern in the network becomes regular (in a regular read-write pattern each processor performs a fixed number of reads and writes per time unit). The second algorithm is TIME-ADAPTIVE-REPLICATION that works in a star network, and converges to a time-optimal replication scheme when the read-write pattern in the network becomes regular. Finally, we proposed the LOWER-BOUND algorithm, that moves the replication scheme in an optimal fashion for each sequence of read-write requests; it needs complete knowledge of future read-write requests.

Acknowledgement

We wish to thank Jeff Ullman and Moti Yung for helpful discussions.

References

[AE1] D. Agrawal and A. El Abbadi, "Storage Efficient Replicated Databases", IEEE Trans. on Data Engineering, 2(3), Sept. 1990.

[AE2] D. Agrawal and A. El Abbadi, "Efficient Techniques for Replicated Data Management", Proc. of the Workshop on Management of Replicated Data, IEEE CS Press, 1990.

[ABG1] R. Alonso, D. Barbara, H. Garcia Molina, "Quasi-copies: Efficient data sharing for information retrieval systems" Proc.

of EDBT'88, LNCS 303, Springer Verlag.

[ABG2] R. Alonso, D. Barbara, H. Garcia Molina, "Data caching issues in an information retrieval system," ACM Transactions on Database Systems, **15** :3, 1990.

[B] L. Belady, "A Study of Page Replacement Algorithms for a Virtual Storage Computer", IBM Systems Journal, 5(2), 1966.

[BG] D. Barbara, H. Garcia Molina, "The case for controlled inconsistency in replicated data," Proc. of the IEEE workshop on replicated data, 1990.

[BHG] P. Bernstein, V. Hadzilacos, and N. Goodman, "Concurrency Control and Recovery in Database Systems", Addison Wesley, 1987.

[BLS] A. Borodin, N. Linial, M. Saks "An Optimal Online Algorithm for Metrical Task Systems", Proc. STOC, 1987.

[BS] D. L. Black and D. D. Sleator "Competitive Algorithms for Replication and Migration Problems", manuscript, 1989.

[CP] S. Ceri and G. Pelagatti, "Distributed Database Principles and Systems," McGraw-Hill, 1984.

[CL] M. Chrobak and L. L. Larmore "An Optimal Online Algorithm for k Servers on Trees", manuscript, 1990.

[DGS] S. Davidson, H. Garcia-Molina, D. Skeen, "Consistency in Partitioned Networks", ACM Computing Surveys, **17** :3, 1985.

[DF] L. W. Dowdy and D. V. Foster, "Comparative Models of the File Assignment Problem", ACM Computing Surveys, **14** :2, 1982.

[E] A. El Abbadi, "Adaptive Protocols for Managing Replicated Distributed Databases", to appear in the Symposium on Parallel and Distributed Processing, 1991.

[GB] H. Garcia-Molina and D. Barbara, "How to assign Votes in a Distributed System", Journal of the ACM, **32** :4, 1985.

[GS] B. Gavish and O. Sheng, "Dynamic File Migration in Distributed Computer Systems", Communication of the ACM, **33** :2, 1990.

[GZ] O. Gerstel and S. Zaks, "A New Characterization of Tree Medians with Applications to Distributed Algorithms", submitted to Networks.

[H] M. Herlihy, "Dynamic Quorum Adjustments for Partitioned Data", ACM Transactions on Database Systems, **12** :2, 1987.

[KMRS] A. Karlin, M. Manasse, L. Rudolph, D. Sleator, "Competitive Snoopy Caching", Algorithmica, **3** :1, 1988.

[KB] N. Krishnakumar and A. Bernstein, "Bounded ignorance in replicated systems," Proc. of ACM-PODS '91.

[LLS] R. Ladin, B. Liskov, L. Shrira, "A technique for constructing highly available distributed services," Algorithmica 3, 1988.

[J] C. Jordan, "Sur les assembblages des lignes", J. Reine Angew. Math., 70 1869.

[OV] M. T. Ozsu and P. Valduriez, "Principles of Distributed Database Systems," Prentice Hall, 1991.

[ST] D. Sleator and R. Tarjan, "Amortized Efficiency of List Update and Paging Rules," CACM 28(2), 1985.

[WM] O. Wolfson and A. Milo, "The Multicast Policy and Its Relationship to Replicated Data Placement", ACM Transactions on Database Systems, **16** :1, 1991.

[Z] B. Zelinka, "Medians and Peripherians on Trees", Arch. Math. (Brno), 1968.

APPENDIX A

In this appendix we show that if, starting at some point in time, the read-write pattern of each processor becomes regular, then, after a convergence period, the (replication scheme of the) CAR algorithm becomes stable.

We start with some definitions. We assume a *synchronous* system, whose operation is partitioned into equal-length time-units. The read-write pattern of a processor, p, in the network becomes *regular* at point in time t, if at each subsequent time-unit p requests the same number of reads, r_p, and the same number of writes, w_p. We assume that in a synchronous system read or write requests that are issued in some time unit, arrive and are serviced within the same time unit. A dynamic replication algorithm can change the replication scheme only at the end of a time unit,

and then the change is instantaneous. A schedule, subschedule, configured scheduled, and a dynamic replication algorithm, in a synchronous system are defined identically to the asynchronous case, except that each read or write operation is mapped to some time-unit, in which it is requested and executed. In a synchronous environment, since the CAR algorithm is restricted to execute the replication-scheme change only at the end of a time unit, we adapt it to *count* operations instead of testing for perfect interleaving. So, the expansion-test of a processor i that is an *R-neighbor* is the following (executed instantaneously at the end of a time-unit):

(Synchronous-Expansion-Test) If during the time unit, the number of read requests made by a neighbor j that is not in R, exceeds the number of write requests made by i or by a neighbor different than j, then i tells j to join R.
The contraction-test of a processor j that is an *R-fringe* is the following:

(Synchronous-Contraction-Test) If during the time unit, the number of write requests made by the single neighbor i that is in R, exceeds the number of read requests received by j, then j exits from R.

The following switch-test is executed by a processor i that is in R but none of its neighbors is in R, provided that the expansion test of i failed:

(Synchronous-Switch-Test) If during the time unit, the number of operations requested by some neighbor n, exceeds the number of all other operations executed by i, then the item is transferred from i to n.

A dynamic replication algorithm is *oscillatory* on some subschedule, if the replication scheme associated with the operations in alternate time units is identical. For example, the replication scheme R is associated with the operations in time-units 1, 3, 5, 7, etc., and the replication scheme S is associated with the operations in time-units 2, 4, 6, 8, etc. Replication scheme R is a *neighbor* of replication scheme S if S can be obtained by adding to R some neighbors of R and removing from R some R-fringe nodes. Clearly, if R is a neighbor of S then S is a neighbor of R. A dynamic replication algorithm is *almost* stable on some subschedule, S, if it is oscillatory, and the two replication schemes associated with the operations in S are neighbors.

Theorem 1: Suppose that the read write pattern of every processor in the network becomes regular starting at time-point t. Denote by c the number of processors in the network. Then after at most c time units, the synchronous CAR algorithm becomes stable or almost stable on a subschedule of arbitrary length.

The proof of theorem 1 will be provided in the full paper.

A similar result can be shown for the synchronous version of the TAR algorithm.

APPENDIX B

This appendix is dedicated to the proof of theorem 2. The proof consists of several lemmas, leading to lemma 4; the lemma provides four properties that characterize an optimal replication scheme. Then, in the proof of theorem 2, we show that when the CAR algorithm stabilizes, the stability scheme possesses these four properties. We will start with some definitions and notations. Intuitively, for a network and a read-write pattern, the median is a node for which the sum of weighted-distances to the other nodes is minimum. Formally, in a tree, let l_{vu} denote the length (in edges) of the simple path between v and u. Denote the set of processors of a subtree, t, of the network T by $V(t)$. A *median* of the network is a node, m, for which $\sum_{i \in V(T)} (\#R_i + \#W_i) \cdot l_{mi}$ is minimum.[5]

Let R be a subset of nodes that induces a connected subgraph of the network. Denote by i a processor that is either an R-fringe, or, is not in R, but is a neighbor of some processor in R. Consider the removal of the edge between i and its neighbor in R, say j. It disconnects the network into two subtrees: a subtree that contains i, that we denote T_{i-R}, and a subtree that contains R, that we denote T_{R-i} (see Fig. 1). The subtrees may also be denoted T_{i-j} and T_{j-i}, respectively.

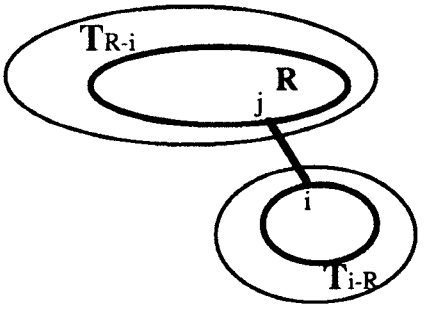


Figure 1: illustration of T_{i-R} and T_{R-i}.

Lemma 1: Let A be a read-write pattern, and let s and t be two neighbors in the tree. Then

[5] The notion of a median was defined independently of a replication scheme, and dates back to 1869 (see [J]; it was further studied in [Z] and [GZ]). The median is a node, m, for which the sum of distances to all the other nodes is minimum, i.e., $\sum_{i \in V(T)} l_{mi}$ is minimum (in contrast, the center of a tree is the node for which the *maximum* of the distances is minimum). In this sense, we actually define the notion of a weighted-median, but since this is the only type of median we discuss, we use the shorter term.

$$\sum_{u \in V(T)} (\#R_u + \#W_u)\cdot l_{su} = \sum_{u \in V(T)} (\#R_u + \#W_u)\cdot l_{tu} + \sum_{u \in V(T_{t-s})} (\#R_u + \#W_u) - \sum_{u \in V(T_{s-t})} (\#R_u + \#W_u).$$

Proof: Observe that by considering separately T_{s-t} and T_{t-s}, and by substituting l_{tu} for l_{su} we obtain:

$$\sum_{u \in V(T)} (\#R_u + \#W_u)\cdot l_{su} = \sum_{u \in V(T_{t-s})} (\#R_u + \#W_u)\cdot (l_{tu} + 1) + \sum_{u \in V(T_{s-t})} (\#R_u + \#W_u)\cdot (l_{tu} - 1).$$

By manipulation of the right hand side of the equation, the lemma follows. []

Lemma 2: Let A be a read-write pattern. A node m is a median if and only if for each neighbor w of m:

$$\sum_{u \in V(T_{m-w})} (\#R_u + \#W_u) \geq \sum_{u \in V(T_{w-m})} (\#R_u + \#W_u).$$

Proof: (==>) Suppose that m is a median, but for one of its neighbors, w,

$$\sum_{u \in V(T_{m-w})} (\#R_u + \#W_u) < \sum_{u \in V(T_{w-m})} (\#R_u + \#W_u).$$

Then, by lemma 1,

$$\sum_{u \in V(T)} (\#R_u + \#W_u)\cdot l_{wu} < \sum_{u \in V(T)} (\#R_u + \#W_u)\cdot l_{mu},$$

contradiction to m being a median.
(<==) Suppose that $d \neq m$ is a median. Consider the path $d=x(1), x(2), \ldots, x(n-1), x(n)=m$. Since for each neighbor w of m:

$$\sum_{u \in V(T_{m-w})} (\#R_u + \#W_u) \geq \sum_{u \in V(T_{w-m})} (\#R_u + \#W_u),$$

in particular:

$$(1) \quad \sum_{u \in V(T_{m-x(n-1)})} (\#R_u + \#W_u) \geq \sum_{u \in V(T_{x(n-1)-m})} (\#R_u + \#W_u)$$

Furthermore, remember that each node performs a nonnegative number of reads and writes. Thus, intuitively, when removing the edge between $x(n-1)$ and $x(n-2)$, rather than the edge between $x(n)$ and $x(n-1)$ as in inequality (1), the operations of $x(n-1)$ move from the right hand side of the inequality to its left hand side. Therefore,

$$\sum_{u \in V(T_{x(n-1)-x(n-2)})} (\#R_u + \#W_u) \geq \sum_{u \in V(T_{x(n-2)-x(n-1)})} (\#R_u + \#W_u)$$

Moreover, by the same argument, for each $n \geq i \geq 2$

$$\sum_{u \in V(T_{x(i)-x(i-1)})} (\#R_u + \#W_u) \geq \sum_{u \in V(T_{x(i-1)-x(i)})} (\#R_u + \#W_u)$$

In particular,

$$\sum_{u \in V(T_{x(2)-x(1)})} (\#R_u + \#W_u) \geq \sum_{u \in V(T_{x(1)-x(2)})} (\#R_u + \#W_u)$$

This, combined with lemma 1, gives:

$$\sum_{u \in V(T)} (\#R_u + \#W_u)\cdot l_{x(2)u} \leq \sum_{u \in V(T)} (\#R_u + \#W_u)\cdot l_{du}$$

By an easy induction on i is can be shown that:

$$\sum_{u \in V(T)} (\#R_u + \#W_u)\cdot l_{x(i)u} \leq \sum_{u \in V(T)} (\#R_u + \#W_u)\cdot l_{x(i-1)u}$$

for each $n \geq i \geq 2$. Therefore,

$$\sum_{u \in V(T)} (\#R_u + \#W_u)\cdot l_{mu} \leq \sum_{u \in V(T)} (\#R_u + \#W_u)\cdot l_{du}$$

Since d is a median, then

$$\sum_{u \in V(T)} (\#R_u + \#W_u)\cdot l_{mu} = \sum_{u \in V(T)} (\#R_u + \#W_u)\cdot l_{du}$$

and consequently m is also a median. []

Lemma 3: Let A be a read-write pattern in which each node performs at least one operation (read or write). Then there are either one or two medians. Furthermore, if there are two medians then they are neighbors in the network.
Proof: Suppose by contradiction that the lemma does not hold. Then there must be two medians, d and m, that are not neighbors. Consider the path $d=x(1), x(2), \ldots, x(n-1), x(n)=m$, for $n \geq 3$. By lemma 2, for each neighbor w of m:

$$\sum_{u \in V(T_{m-w})} (\#R_u + \#W_u) \geq \sum_{u \in V(T_{w-m})} (\#R_u + \#W_u)$$

In particular:

$$\sum_{u \in V(T_{m-x(n-1)})} (\#R_u + \#W_u) \geq \sum_{u \in V(T_{x(n-1)-m})} (\#R_u + \#W_u)$$

Furthermore, since each node performs at least one operation, for each $n-1 \geq i \geq 2$

$$\sum_{u \in V(T_{x(i)-x(i-1)})} (\#R_u + \#W_u) > \sum_{u \in V(T_{x(i-1)-x(i)})} (\#R_u + \#W_u)$$

In particular,

$$\sum_{u \in V(T_{x(2)-x(1)})} (\#R_u + \#W_u) > \sum_{u \in V(T_{x(1)-x(2)})} (\#R_u + \#W_u)$$

By lemma 2, this contradicts the fact that d is a median. □

Lemma 4: Let A be a read-write pattern in which each processor performs at least one operation (read or write). A replication scheme, R, that satisfies the following four conditions is optimal for A: 1. R is a (connected) subtree, and 2. each node i that is a neighbor of R satisfies: $\sum_{j \in V(T_{i-R})} \#R_j \leq \sum_{j \in V(T_{R-i})} \#W_j$, and 3. Suppose that R contains more than one node and let i be an R-fringe node. Then, $\sum_{j \in V(T_{i-R})} \#R_j \geq \sum_{j \in V(T_{R-i})} \#W_j$. 4. If R is a singleton set, then it consists of a median of the network.

Proof: In [WM] we have shown that the following algorithm computes an optimal replication scheme, RS[6].

TREE-RS [T(V,E), A]: / Algorithm for finding the optimal residence set, given a tree T and a read-write pattern A in which each processor performs at least one read and at least one write */*

1. *init*: color all the processors of V red; initialize RS to a median, m.
2. while there exists a processor in RS with at least one red neighbor, i, do:
3. if $\sum_{j \in V(T_{i-RS})} \#R_j \geq \sum_{j \in V(T_{RS-i})} \#W_j$ then add i to RS; else color i blue.
4. end while.
5. *output*: RS.

The main idea of the proof is to show that if a replication scheme, R, satisfies the four conditions of the lemma, then its cost is equal to the cost of the replication scheme RS (produced by the algorithm *TREE-RS*). The proof proceeds in three stages. In stage I we will show that R contains a median. In stage II we will show that if R and RS are disjoint, then each one of the sets is a singleton, implying that each set consists of a median, and then, based lemmas 2 and 3 the costs are equal. In stage III we show that if R and RS are not disjoint, then actually R is contained in RS, and the two costs are equal.

(Stage I) Suppose that R does not contain a median. Then, by condition 4 it is not a singleton. We will show that conditions 3 and 4 cannot both be satisfied. Let the shortest path between a median and R be $m, n, \cdots, l, k$, where $k \in R$. Since m is a median and each node performs at least one operation,

$$\sum_{j \in V(T_{l-R})} (\#W_j + \#R_j) > \sum_{j \in V(T_{R-l})} (\#W_j + \#R_j)$$

By condition 2 of the lemma

$$\sum_{j \in V(T_{l-R})} \#R_j \leq \sum_{j \in V(T_{R-l})} \#W_j.$$

Together, the last two inequalities imply that

(0) $$\sum_{j \in V(T_{l-R})} \#W_j > \sum_{j \in V(T_{R-l})} \#R_j.$$

Since R is not a singleton, let g be an R-fringe that is different than k. It is easy to see that $V(T_{g-R})$ is a proper subset of $V(T_{R-l})$ and $V(T_{l-R})$ is a proper subset of $V(T_{R-g})$. Therefore, based on inequality (0)

$$\sum_{j \in V(T_{R-g})} \#W_j > \sum_{j \in V(T_{g-R})} \#R_j$$

which in turn contradicts condition 3 of the lemma.

(Stage II) The set R contains a median by Stage I, and the set RS contains a median by step 1 of the algorithm *TREE-RS*. Assume that R and RS are disjoint. Consequently, each one of the two sets contains a different median. Let the two medians be m and n, and suppose without loss of generality that $m \in RS$ and $n \in R$.

Suppose, by way of contradiction, that RS contains more than one node. Let k be a node of RS that is different than m, and is a leaf of the subtree induced by RS. We will show that the cost of RS can be reduced by replacing k by n, contradicting the minimality of $cost(RS, A)$. Let us consider how this change will affect the cost of RS. For this purpose we will partition the nodes of the tree network into three disjoint subsets, illustrated in Fig. 2: set-1 consisting of the nodes of T_n, set-2 consisting of the nodes of T_m that are not in T_{k-RS}, set-3 consisting of and the nodes of T_{k-RS}.

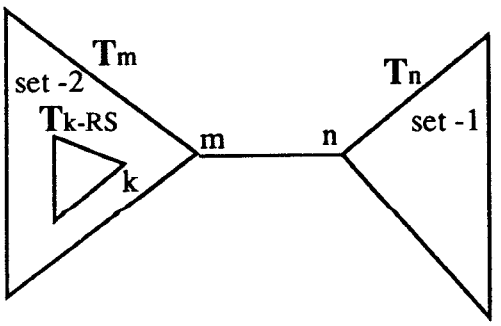


Figure 2:

[6] Actually, in [WM] we have proven that RS is an optimal replication scheme, provided that A is a read-write pattern in which each processor performs at least one read and at least one write. However, the same proof works for a read-write pattern in which each processor performs at least one operation.

The cost of each read originating in set-1 will decrease by one (since it will access the replica on n instead of m), and the cost of each write originating in set-1 will decrease by one (since it will not have to access k). The cost of each read originating in set-2 remain the same (since it will access m), and the cost of each write originating in set-2 will remain the same (since it will access m instead of k). The cost of each read originating in set-3 will increase by one (since it will access the neighbor of k in RS, instead of k), and the cost of each write originating in set-3 will increase by one (since it will have to access n). Since m and n are medians, and T_{k-RS} is a proper subset of T_m, there are strictly more operations originating in set-1 than in set-3. Consequently, the replacing k by n will reduce $cost(RS, A)$.

Suppose now, again by way of contradiction, that R contains more than one node. Let k be a node of R that is different than m, and is a leaf of the subtree induced by R. We will show that the four conditions in the lemma cannot be simultaneously satisfied. For this purpose we will consider three subtrees, T_{m-n}, T_{n-m} and T_{k-R}, illustrated in Fig. 3. Note that T_{k-R} is included in T_{n-m}.

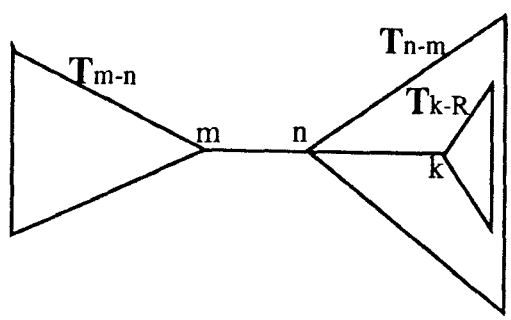


Figure 3:

By condition 2 of the lemma,

(1) $\sum_{j \in V(T_{m-n})} \#R_j \leq \sum_{j \in V(T_{n-m})} \#W_j$

(m is the neighbor of R).

By condition 3 of the lemma,

(2) $\sum_{j \in V(T_{m-n})} \#W_j + \sum_{j \in V(T_{n-m})-V(T_{k-R})} \#W_j \leq \sum_{j \in V(T_{k-R})} \#R_j$

(k is the R-fringe node).

Note that the left hand side of inequality (2) is actually $\sum_{j \in V(T_{R-k})} \#W_j$.

If there is any write issued in some node of $T_{n-m}-T_{k-R}$ (i.e. in a node that is in T_{n-m} but not in T_{k-R}), then, by inequality (2), $\sum_{j \in V(T_{m-n})} \#W_j < \sum_{j \in V(T_{k-R})} \#R_j$, and this, combined with inequality (1), implies that

$$\sum_{j \in V(T_{m-n})} (\#R_j + \#W_j) < \sum_{j \in V(T_{n-m})} (\#R_j + \#W_j)$$

(since T_{k-R} is a subset of T_{n-m}). But, by lemma 2, this contradicts the fact that m is a median. Thus, $\sum_{j \in V(T_{n-m})-V(T_{k-R})} \#W_j = 0$. Since n is in $V(T_{n-m})-V(T_{k-R})$, and it performs at least one operation, this operation must be a read. Combined with (2), this implies that

(3) $\sum_{j \in V(T_{m-n})} \#W_j < \sum_{j \in V(T_{n-m})} \#R_j$.

By inequalities (1) and (3)

$$\sum_{j \in V(T_{m-n})} (\#R_j + \#W_j) < \sum_{j \in V(T_{n-m})} (\#R_j + \#W_j)$$

But this contradicts the fact that m is a median.

(Stage III) Suppose that R and RS are not disjoint. We will show first that R is contained in RS. Suppose that there is a node of R that is not in RS. Since RS induces a connected subtree of the network, there must be leaf, q of the subtree induced by R, that is not in RS (see Fig. 4)

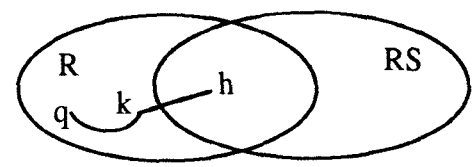


Figure 4:

Consider the shortest path $q=x_1, x_2, \cdots, x_{n-1}=k, x_n=h$ from q to a node h in $R \cap RS$. During the construction of RS by TREE-RS, the node k must have been examined in step 3. Since k was not added to RS, $\sum_{j \in V(T_{k-RS})} \#R_j < \sum_{j \in V(T_{RS-k})} \#W_j$. Then, for every $n-1 \leq i \leq 1$:

$$\sum_{j \in V(T_{x(n-i)}-x(n-i+1))} \#R_j < \sum_{j \in V(T_{x(n-i+1)}-x(n-i))} \#W_j.$$

In other words, for every i, if we remove the edge between $x(i)$ and $x(i+1)$, then there are more writes issued in the subnetwork that contains $x(i+1)$ than there are reads issued in the subnetwork that contains $x(i)$. This is true in particular for $i=1$. But then, since q is an R-fringe node, this contradicts condition 4 of the lemma.

Finally, we will show that if $R \subseteq RS$ then the costs of the two sets are equal. This claim is obviously true is $R = RS$, therefore suppose that $R \subset RS$. We will show that all the nodes of

$RS - R$ can be added to R without changing the cost of R, and this will conclude the proof.

Let k be a node in $RS - R$ that is a neighbor of R (see Fig. 5).

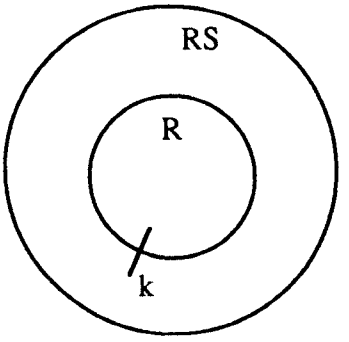


Figure 5:

By condition 3 of the lemma, we know that

(4) $$\sum_{j \in V(T_{k-R})} \#R_j \leq \sum_{j \in V(T_{R-k})} \#W_j.$$

We will show that actually:

(5) $$\sum_{j \in V(T_{k-R})} \#R_j = \sum_{j \in V(T_{R-k})} \#W_j.$$

Suppose that all the nodes of $V(T_{k-R})$ that are in RS are removed from RS. This will affect the cost of RS as follows. The cost of all the reads performed in $V(T_{k-R})$ will increase by one, and the cost of all the writes performed in $V(T_{R-k})$ will decrease by one; the cost of any other operation will not change. Since the cost of RS is minimum,

(6) $$\sum_{j \in V(T_{k-R})} \#R_j \geq \sum_{j \in V(T_{R-k})} \#W_j.$$

By combining (4) and (6) we obtain (5). Equation (5) implies that the addition of k to R will not change the cost of R, for the following reason. This addition will increase the cost of each write performed in $V(T_{R-k})$ by one, and will decrease the cost of each read performed in $V(T_{k-R})$ by one; the cost of any other operation will not change. This proof can be repeated verbatim, if after the addition of k to R the set $RS - R$ is not empty. □

Proof of Theorem 2: We will show that either R is optimal for A, or, there is a replication scheme R', that is obtained by applying a transformation to R, such that R' satisfies: 1. it is optimal for A, and 2. $cost(R,A) - cost(R',A) \leq c$. The initial replication scheme, consisting of the whole set of processors, clearly induces a connected subgraph of the network. Since the COST-ADAPTIVE-REPLICATION algorithm only adds neighbors of the current replication scheme and drops fringe nodes, R induces a connected subgraph of the network. If R also satisfies conditions 2, 3, and 4 of lemma 4, then it is optimal for A, and the proof is complete. Otherwise, we will show that the conditions of lemma 4 are satisfied by the replication scheme R', that is obtained from R by the application of a transformation, called *Transform*. It consists of the addition of some neighbors, and removal of some R-fringe nodes.

Description of *Transform*:

Suppose that condition 2 of lemma 4 is not satisfied for R. Then we execute the following, step 1 of *Transform*. There is at least one neighbor k of R for which:

(10) $$\sum_{j \in V(T_{k-R})} \#R_j > \sum_{j \in V(T_{R-k})} \#W_j.$$

Transform calls k an R-add node. Now we will make an important observation about k. Denote by i the neighbor of k that is in R. Remember that i performs the expansion test during the operations in the schedule S, and since the CAR algorithm is stable on S, we know that i does not change the replication scheme. Since inequality (10) holds, the sequence of requests at node i during the schedule S, when considering only the reads from $V(T_{k-R})$ and writes from $V(T_{R-k})$, is the following: $q = r,w,r,.....,r,w,r$. In other words, the sequence q starts and ends with a read from $V(T_{k-R})$, and the number of such reads exceeds the number of writes from $V(T_{R-k})$ by one.

Suppose that condition 3 of lemma 4 is not satisfied for R. Then we execute the following, step 2 of *Transform*. There is at least one R-fringe node, k, for which:

(11) $$\sum_{j \in V(T_{k-R})} \#R_j < \sum_{j \in V(T_{R-k})} \#W_j.$$

Transform calls k an R-drop node. Now we will make the following observation about k. It performs the contraction test during the operations in the schedule S, and since the CAR algorithm is stable on S, we know that k does not change the replication scheme. Since inequality (11) holds, the sequence of requests at node k during the schedule S, when considering only the reads from $V(T_{k-R})$ and writes from $V(T_{R-k})$, is the following: $q = w,r,w,.....,r,w$. In other words, the sequence q starts and ends with a write from $V(T_{R-k})$, and the number of such writes exceeds the number of reads from $V(T_{k-R})$ by one.

Suppose that condition 4 of lemma 4 is not satisfied for R. If step 1 of *Transform* defined any R-add nodes, then we go step 4 of *Transform*. Otherwise we execute the following, step 3 of *Transform*. R is a singleton, say $\{k\}$. There is one neighbor of k, n, for which during S the number of operations requested by n is bigger than the number of operations initiated at some node of $V(T_{k-n})$. *Transform* calls k an R-drop node, and n and R-add node. Now we will make the following observation about k and n. k performs the switch and expansion tests during the operations in the schedule S, and since the CAR algorithm is stable on S, we know that k does not change the replication scheme. Then the sequence of requests at node k during the schedule S is the following: $q = a,b,a,.....,b,a$, where the a's are operations issued by n, and the b's are operations initiated at some node of

$V(T_{k-n})$. In other words, in the sequence q the number of a's exceeds the number of b's by one.

Finally, in step 4 of *Transform* we define R' as follows. If the set

$$(R \cup \{all\ R\text{-}add\ nodes\}) - \{all\ R\text{-}drop\ nodes\}$$

is nonempty, let us call it R'. Otherwise R must consist of two nodes, say k and m, both of which are R-drops. Then, let us define $R' = \{k\}$.

This completes the description of *Transform*.

Now observe that the removal of each R-drop node from R decreases $cost(R,A)$ by one (by the definition of an R-drop node). Similarly, the addition of each R-add node to R decreases $cost(R,A)$ by one (by the definition of an R-add node). To complete the proof of the theorem, left to prove is that R' is optimal, and we will do so by showing that the four conditions of lemma 4 are satisfied for R'.

(Case 1.) Suppose that R' contains more than one node. It can be deduced from the definitions that an R-add and an R-drop cannot be neighbors. Thus, R' is connected. To show that condition 2 of lemma 4 is satisfied for R', consider a node i that is a neighbor of R'. If i is also a neighbor of R, then it obviously satisfies: $\sum_{j \in V(T_{i-R'})} \#R_j \leq \sum_{j \in V(T_{R'-i})} \#W_j$. Otherwise, i is either a neighbor of an R-add node, or i is an R-drop node. It can be verified from the definition of an R-add node, and the fact that each node performs at least one operation, that in both cases: $\sum_{j \in V(T_{i-R'})} \#R_j \leq \sum_{j \in V(T_{R'-i})} \#W_j$. Thus condition 2 of lemma 4 is satisfied for R'.

To show that condition 3 of lemma 4 is satisfied for R', consider a node i that is an R'-fringe node. If i is also an R-fringe, then it obviously satisfies: $\sum_{j \in V(T_{i-R'})} \#R_j \geq \sum_{j \in V(T_{R'-i})} \#W_j$. Otherwise, i is either an R-add node, or i is in R and it becomes an R'-fringe node as a result of being a neighbor of some R-drop node. In the first case it can be verified from the definition of an R-add node, that: $\sum_{j \in V(T_{i-R'})} \#R_j \geq \sum_{j \in V(T_{R'-i})} \#W_j$. In the second case this inequality holds by the definition of an R-fringe node, and by the fact that each node performs at least one operation. Thus condition 3 of lemma 4 is satisfied for R'.

(Case 2.) Suppose that R' is a singleton set, $\{k\}$. We have to show that conditions 2 and 4 of lemma 4 are satisfied.
(Case 2.1) Suppose that R is also a singleton set, $\{m\}$.
(Case 2.1.1) Suppose that $R' = R$. Then, since k performs the switch test, and since R is the stability scheme on S, and since *Transform* does not change R, k is a median. Since k performs the expansion-test, and since no R-add nodes were defined by *Transform*, condition 2 of lemma 4 is satisfied for R'.
(Case 2.1.2) Suppose that $R' \neq R$. Then k is a unique R-add and m is a unique R-drop. By the way these were defined by *Transform*, it is easy to see that condition 4 of lemma 4 is satisfied for $\{k\}$. Condition 2 of lemma 4 is satisfied for the following reason. Since the expansion test performed by m does not change R, then:
(12)
$$\sum_{j \in V(T_{k-m})} \#R_j \leq \sum_{j \in V(T_{m-k})} \#W_j + 1.$$
Consider l, a neighbor of k that is different than m. Since $V(T_{k-l}) \subset V(T_{m-k})$, and since k performs at least one operation, then: $\sum_{j \in V(T_{l-k})} \#R_j \leq \sum_{j \in V(T_{k-l})} \#W_j$. Consider m. Since $R \neq R'$ and both are singletons,
(13)
$$\sum_{j \in V(T_{k-m})} \#O_j \leq \sum_{j \in V(T_{m-k})} \#O_j + 1,$$
where $\#O_j$ denotes the number of operations (read or write) issued at node j. Combining (12) and (13) we obtain: $\sum_{j \in V(T_{m-k})} \#R_j \leq \sum_{j \in V(T_{k-m})} \#W_j$.
(Case 2.2) Suppose that R is not a singleton set (i.e. R' is obtained by deleting nodes from R). It can be shown that condition 2 of lemma 4 is satisfied for $\{k\}$, by the same line of reasoning as used in case 1. For showing that condition 4 is satisfied, we will analyze three subcases.
(Case 2.2.1) Suppose that R consists of two nodes: k, and another node m, that is an R-drop. Since k is not an R-drop,
(14)
$$\sum_{j \in V(T_{k-m})} \#R_j \geq \sum_{j \in V(T_{m-k})} \#W_j.$$
Since m is an R-drop,
(15)
$$\sum_{j \in V(T_{k-m})} \#W_j = \sum_{j \in V(T_{m-k})} \#R_j + 1.$$
Based on (14) and (15), $\sum_{j \in V(T_{k-m})} \#O_j > \sum_{j \in V(T_{m-k})} \#O_j$, and therefore condition 4 of lemma 4 is satisfied for neighbor m that is an R-drop.
Now consider another neighbor of k, say g (g is not in R since R consists of two nodes, and it is also not an R-add since $R' = \{k\}$. Since g is not an R-add,
(16)
$$\sum_{j \in V(T_{g-k})} \#R_j \leq \sum_{j \in V(T_{k-g})} \#W_j.$$
Since m is an R-drop, $\sum_{j \in V(T_{k-m})} \#W_j = \sum_{j \in V(T_{m-k})} \#R_j + 1$. Additionally, $V(T_{g-k}) \subset V(T_{k-m})$, and $V(T_{m-k}) \subset V(T_{k-g})$, and k performs at least one operation. Consequently,
(17)
$$\sum_{j \in V(T_{g-k})} \#W_j \leq \sum_{j \in V(T_{k-g})} \#R_j.$$
Based on (16) and (17) condition 4 of lemma 4 is satisfied for g.

(Case 2.2.2) Suppose that R consists of two nodes: k and m, both of which are R-drops. Since m is an R-drop, equation (15) holds, and since k is an R-drop

(18)
$$\sum_{j \in V(T_{m-k})} \#W_j = \sum_{j \in V(T_{k-m})} \#R_j + 1.$$

Based on (15) and (18),

(19)
$$\sum_{j \in V(T_{m-k})} \#O_j = \sum_{j \in V(T_{k-m})} \#O_j.$$

Based on (19) it is easy to see that for any other neighbor of k, say l, $\sum_{j \in V(T_{k-l})} \#O_j \geq \sum_{j \in V(T_{l-k})} \#O_j$. Thus condition 4 of lemma 4 is satisfied for R'.

(Case 2.2.3) Suppose that R consists of k plus two or more nodes, all of which are R-drops. Consider an R-drop neighbor of k, say m. Equation (15) holds for m. Now consider another R-drop neighbor of k, say n.

(20)
$$\sum_{j \in V(T_{k-n})} \#W_j = \sum_{j \in V(T_{n-k})} \#R_j + 1.$$

Since $V(T_{n-k}) \subset V(T_{k-m})$, and since each node performs at least one operation,

(21)
$$\sum_{j \in V(T_{m-k})} \#W_j \leq \sum_{j \in V(T_{k-m})} \#R_j.$$

Based on (15) and (21), $\sum_{j \in V(T_{k-m})} \#O_j > \sum_{j \in V(T_{m-k})} \#O_j$. Thus condition 4 of lemma 4 is satisfied for each R-drop node. If there are other neighbors of k, that are not R-drops, the proof that for such a neighbor, say g, $\sum_{j \in V(T_{k-m})} \#O_j \geq \sum_{j \in V(T_{m-k})} \#O_j$ is identical to the one in case 2.2.1. □