THE MULTICAST POLICY AND ITS RELATIONSHIP TO REPLICATED DATA PLACEMENT

Amir Milo, Ouri Wolfson

Columbia University
Dept. of Computer Science

Technical Report CUCS-462-89

The Multicast Policy and Its Relationship to Replicated Data Placement[1]

Amir Milo
Department of Computer Science
The Technion - Israel Institute of Technology
Haifa 32000, Israel

Ouri Wolfson
Department of Computer Science
Columbia University
New York, NY 10027.

ABSTRACT

In this paper we consider the communication complexity of maintaining the replicas of a logical data-item, in a database distributed over a computer network. We propose a new method, called the minimum spanning tree write, by which a processor in the network should multicast a write of a logical data-item, to all the processors that store replicas of the item. Then we show that the minimum spanning tree write is optimal from the communication cost point of view. We also demonstrate that the method by which a write is multicast to all the replicas of a data-item, affects the optimal replication scheme of the item, i.e., at which processors in the network the replicas should be located. Therefore, next we consider the problem of determining an optimal replication scheme for a data item, assuming that each processor employs the minimum spanning tree write at run-time. The problem for general networks is shown NP-Complete, but we provide efficient algorithms to obtain an optimal allocation scheme for three common types of network topologies. They are completely-connected, tree, and ring networks. For these topologies, efficient algorithms are also provided for the case in which reliability considerations dictate a minimum number of replicas.

Categories and Subject Descriptors: H.2.4 [Database Management]: Transaction Processing; C.4 [Computer Systems Organization]: Performance of Systems; C.2.4 [Computer-Communication Networks]: Distributed Systems; D.2.8 [Software Engineering]: Metrics

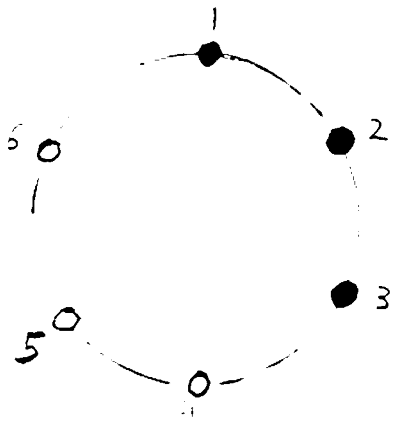
General Terms: Algorithms, Performance, Theory

Additional Key Words and Phrases: File allocation, Replicated data, Complexity, NP-Complete, Computer network, Message passing.

[1] This research was supported in part by DARPA research grant #F-29601-87-C-0074, and by the Center for Advanced Technology at Columbia University under contract NYSSTF-CAT(89)-5.

1. Introduction

Consider a replicated database that is distributed among the processors in a communication network. We model the communication network by a connected undirected graph; the nodes represent processors, and the edges represent two-way communication links. A logical data-item is a file, a relation, or part of a relation, and each such item is physically replicated at one or more processors in the network. When a transaction writes the logical data-item, the write has to be propagated to all its physical replicas. The first question we address in this paper is: what is the optimal write-multicast-policy, i.e., way of multicasting a write of a logical data-item to all its physical replicas? By an optimal write-multicast-policy, or simply an optimal write-policy, we mean a propagation scheme that puts the minimal load on the communication network.



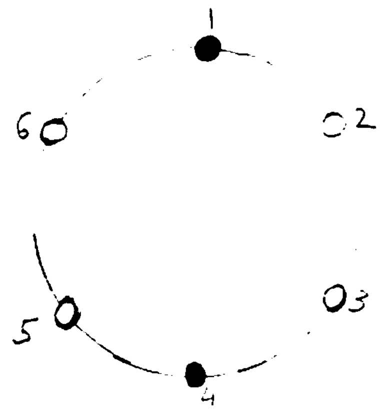


figure 1.1: A communication network. The shaded processors store replicas of a data-item.

For example, assume that six processors are interconnected in a ring, as in figure 1.1a. The shaded processors represent the residence set of a data-item, i.e., the processors that store replicas of the data-item. Suppose that processor 6 is the writer, i.e., has to send the data-item to all the processors of the residence set. The most efficient way for processor 6 to propagate the write is the following. It sends the data-item to processor 1, which then sends the data-item to processor 2; processor 2, in turn, sends the data-item to processor 3. Overall, the data item traverses three communication links. This is clearly a lower load on the communication network than the one put by the write policy that we call *naive*, in which processor 6 propagates the write in three messages: from 6 to 1, from 6 to 2, and from 6 to 3. In this case the data item traverses 1+2+3 communication links.

In this paper we first establish the optimal write-policy for an arbitrary communication network, an arbitrary residence set, and an arbitrary writer processor. It is the policy that we call the *minimum−spanning−tree* write, in which the writer propagates the data-item along the edges of a minimum spanning three of the distance graph; it is a complete weighted graph in which the nodes, or the participants, are the writer and the processors of the residence set, and each edge represents the distance in the communication network between every pair of participants. When using the minimum-spanning-tree write-policy, processor 6 in figure 1.1a will indeed propagate the logical data-item to processors 1, 2, and 3 in the efficient way suggested above.

Of course, efficient multicast algorithms have been studied previously[2] (e.g. [CD, CZ, De, D, DM, P]). However, the following question, that arises in the database context, has not been addressed. Does the write policy used by the writers affect the optimal residence set, i.e., the choice of the residence set, for which the total load on the communication network is minimized? The answer is yes. For example, consider the ring network of figure 1, and assume that each processor performs two reads and one write of the logical data-item, per time unit. Each read is, expectedly, performed from the closest (in terms of links-distance in the network) processor of the residence set. If each processor propagates each write that it performs using the minimum-spanning-tree policy, then the optimal residence set is the one illustrated in figure 1.1a (obviously, any string of three processors is optimal). The total number of transmissions of the data-item along a communication link is 24 per time unit. On the other hand, if each processor uses the naive-write policy, as the previous works have assumed, then the optimal residence set is the one in figure 1.1b (26 link-transmissions per time unit). Moreover, the residence set of figure 1.1a is not optimal for the naive-write policy (35 link-transmissions), and the residence set of figure 1.1b is not optimal for the minimum-spanning-tree write policy (26 link-transmissions).

Therefore, after establishing that the minimum-spanning-tree write-policy minimizes communication at run time, we assume that each processor uses it, and we examine the optimal residence set problem. It is the problem of determining where the replicas of a data-item should be placed, in order to minimize the

[2] The underlying assumptions are often different than ours, since they did not assume a distributed database environment as we do. For example in [DM] each processor in the network knows its own identity, but not necessarily the identity of its neighbors. In contrast, notice that a processor usually does know at which processors the database is distributed.

communication load. We take what Ceri et. al. call the user viewpoint ([CMP]), by assuming that the network topology, and the read/write activity at each processor, are predefined.

The optimal residence set problem, also called the file-allocation problem, has been studied extensively in the literature (see [DF] for a survey). Most existing works attempt to minimize the communication cost, as well as other parameters, such as storage costs ([C, ML]), communication channels capacity ([MR]), or the communication network topology ([IK]). Concentrating on the communication cost alone makes our model relatively simple. Additionally, and more importantly, all the previous works that we are aware of, have assumed a naive write policy (e.g., [C, DF, FH, MIMH, RW]). The naive-write policy assumption is implied by the formulation of the total communication cost of a multicast transmission, as simply the sum of the communication costs between the sender and each one of the receivers. As the above example demonstrates, when using the minimum-spanning-tree write, the cost of a multicast from processor 6 to processors 1, 2, and 3 is not the sum 1+2+3. The observation that the write policy affects the optimal residence set, leads us to believe that the optimal residence set problem must be reconsidered from this new angle, i.e. assuming other write-multicast policies. Since the optimal residence set is determined with the purpose of minimizing the communication cost, it makes sense to assume that each processor minimizes the communication cost at run-time.

In this paper we determine that when the minimum spanning tree write is used, the problem of finding an optimal residence set is NP-complete for general communication networks. However, our simple problem formulation enables us to provide efficient algorithms for solving the problem in the following widely used network topologies: completely connected, tree and ring. Moreover, for general networks, the problem is NP-complete even if the number of reads of the data item is identical at all the processors, and so is the number of writes (the balanced-load case). On the other hand, for the special topologies the problem can be solved efficiently, even if different processors have different access patterns to the data-item (the unbalanced-load case).

So far we discussed replicated data from the performance point of view. However, one of the main purposes for replicating data in computer networks is improved reliability; if some, but not all, of the processors of the residence set fail, the applications should still be able to access the data-item. Therefore, often the communication cost has to be optimized subject to the constraint that the number of replicas in the network is not lower than some threshold, t. This threshold is calculated a priori by means outside the scope of this paper, such as the failure probability of a processor. Consequently, for the special topologies, i.e., completely-connected, tree, and ring, we also provide efficient algorithms for the the problem of finding the t-reliable optimal residence set, i.e. the optimal residence set in which the number of replicas is at least t. We do so for the balanced- and unbalanced-load cases.

The rest of the paper is organized as follows. In section 2 we establish that the minimum-spanning-tree write policy achieves the minimum communication cost. The optimal residence set problem is defined and shown in *NP−Complete* for general networks, in section 3. In section 4 we present the positive results, i.e., the algorithms that solve the optimal residence set problem for completely-connected, tree, and ring networks. In section 5 we discuss the effect that reliability considerations have on the results presented in section 4. In section 6 we conclude, and discuss future work.

2. Read and Write Message-Costs.

Read and write operations for logical entities (or data-items) are issued at each processor in a computer-network. Each such operation is eventually translated into zero or more information messages transmitted in the network; they carry the data-item to or from the processors storing the physical replicas of the logical data-item. In this section we establish the minimal number of network messages required for an arbitrary read and write by a processor. The read case is simple, the write case slightly more involved. We start with some definitions. A *communication network*, or *network* for short, is an undirected connected graph, $G=(V,E)$. V represents a set of processors, and an edge in the network between processors v and w represents a bidirectional communication link between them. Given a network we define a *residence set* to be a subset of V. It represents the processors where some arbitrary fixed data-item is replicated. We assume that reading of a data-item by a processor is implemented by transferring the closest replica to it. Therefore, for a given network and residence set, the *read cost* of a processor v, denoted r_v, is the length (in edges) of the shortest path in the network between v and a processor of the residence set. It represents the number of information messages required for the data-item transfer. Obviously, if v is in the residence set, then the read cost is zero.

Next we establish the write cost for a processor, given a residence set, R. We assume that processor $v \in V$ writes the logical data-item, and call it the *writer*; it may or may not belong to the residence set. The processors in $R \cup \{v\}$ are the *participants* in the write protocol, and are denoted by P. A *write instance* is a directed graph, $I=(P,A)$. Each arc of A represents a replica transfer between two participants, and its cost is the shortest path in the network between the processors at its endpoints. Since the data-item sent by the writer reaches every other participant, we require that there is a path in I from v to each processor in R. The *write−instance cost* is the total cost of its arcs. We are interested in establishing the minimal cost of a write instance. Clearly, for this purpose the only instances to be considered are acyclic. A possible algorithm which sends exactly the messages of a given acyclic instance, I, is the following (assuming I is known to all participants): the writer sends the data-item to its sons, and each processor after receiving the data-item forwards it to its sons and so on, until the data-item reaches the leaves of I.

For the next proposition we need to define the *distance graph*, $D_G(\hat{V})$, for an arbitrary subset of processors $\hat{V} \subseteq V$. $D_G(\hat{V})$ is a complete weighted graph with the set of nodes $\hat{V}$. The weight of an edge between j and k in $D_G(\hat{V})$ is the length of the shortest path in G between j and k. Denote by $mst(D_G(\hat{V}))$ a minimum-spanning-tree of $D_G(\hat{V})$.

Proposition 2.1: Let G be a network, R a residence set, and v a writer. Denote the set $R \cup \{v\}$ by P. Then the necessary and sufficient cost of a write instance is the total weight of $mst(D_G(P))$.

Proof : (necessary) By the definition of a write instance the underlying graph of any write instance is a connected subgraph of $D_G(P)$ that spans all the processors of P. The minimal weight of such a graph is the weight of $MST(D_G(P))$.

(sufficient) Given an $MST(D_G(P))$ we can build the required write instance whose cost is equal to the weight of $MST(D_G(P))$, by directing the edges of the $MST(D_G(P))$ to form a rooted-tree, rooted at the writer. $\square$

Therefore, given a residence-set, R, we define the *write cost* of processor v, denoted w_v, to be the total weight of $mst(D_G(P))$. Denote by d_v, the length of the shortest path in G between v and (some processor of) R. The next lemma will be used extensively in our proofs.

Lemma 2.1: Let $G=(V,E)$ be a network, R a residence set, and $i \in V$ a processor. If R induces a connected subgraph of G, then $w_i=d_i+|R|-1$.

Proof: We prove first that any spanning tree of $D_G(R \cup \{i\})$ is of weight at least $d_i+|R|-1$. The $|R|+1$ nodes of the distance graph require $|R|$ edges in order to form a spanning tree. The lightest edge that is connected to i in $D_G(R \cup \{i\})$ is obviously of weight d_i. Each of the other $|R|-1$ edges is at least of weight one, therefore there is no spanning tree whose weight is less than $d_i+|R|-1$.

Now we prove that a spanning tree of such weight exists. R induces a connected subgraph, thus, there are $|R|-1$ edges of weight one, and this set of edges spans the nodes of R in the distance graph. Also, there exists an edge of weight d_i between node i and some node of R. Therefore there exists a spanning tree whose weight is $d_i+|R|-1$. $\square$

3. The residence set problem

In this section we define the residence set problem, namely the problem of placing the replicas of a given data item to minimize overall message traffic in a computer-network. Then its complexity is established. We assume that the transaction processing load is balanced in the following sense. The number of data-item read operations per time unit, #R, is equal at all the processors, and the same holds for the number of write operations per time unit, #W. We denote the ratio #R/#W by α. In other words, for each write there are α reads at each processor. We assume that α is well defined (i.e. #W > 0), and positive. Given a residence set, R, and α, the *residence set cost*, denoted $cost(R)$, is defined as $\sum_{i \in V} w_i + \alpha \cdot \sum_{i \in V} r_i$. We shall refer to the first sum in the expression as the *total-write-cost*, and to the last sum multiplied by α, as the *total-read-cost*. Intuitively, $\sum_{i \in V} w_i + \alpha \cdot \sum_{i \in V} r_i$ represents the total communication cost when R is the residence set and the time unit is chosen such that #W = 1. (or, it can be looked at as $\#W \cdot \sum_{i \in V} w_i + \#R \cdot \sum_{i \in V} r_i$ divided by #W). We would like to find an *optimal residence set*, i.e. a residence set with the minimal cost. The *residence set* problem, denoted RS, is defined as follows:

Input: A communication network graph $G=(V,E)$, and two positive real numbers, α and C.

Question: Is there a residence set $R \subseteq V$, such that $\sum_{i \in V} w_i + \alpha \cdot \sum_{i \in V} r_i < C$?

Theorem 3.1: *RS* is *NP-Complete*.

Proof : See Appendix.

Although the problem is *NP –Complete* in general, for certain input parameters it can be solved efficiently.

Theorem 3.2: Let $G = (V,E)$ be a network, and assume that the read-write ratio, α, is bigger than $|V|-1$. Then there is a unique optimal residence set, and it is the set of all processors.

Proof : Suppose that $R \subset V$ is an optimal residence set. Let k be a processor in $V-R$, which has a neighbor in R. It is easy to see that $cost(R \cup \{k\}) < cost(R)$, as follows. The total read-cost for $R \cup \{k\}$ is lower than the total read-cost for R, by at least α. For each $i \neq k$, w_i for $R \cup \{k\}$ is higher than w_i for R, by at most one. Therefore, the total write-cost for $R \cup \{k\}$ is higher than the total write cost for R, by at most $|V|-1$. Overall, since $\alpha > |V|-1$, $cost(R) > cost(R \cup \{k\})$, contradicting the optimality of R. □

In the next section we show that for certain common network topologies the problem can be solved efficiently for any read-write ratio.

4. Special topologies

In this section we present the positive results concerning completely-connected, tree, and ring networks. Each topology is treated in its own subsection, in which we provide an algorithm for finding the optimal residence set, and prove its correctness. The algorithm deals with the balanced-load case. At the end of each subsection we discuss the extension of the algorithm to efficiently solve the problem if the load is unbalanced. This is the case in which processor i in the network performs $\#R_i>0$ reads per time-unit, and $\#W_i>0$ writes per time unit. For different processors, i and j, $\#R_i$ may be different than $\#R_j$, and similarly, $\#W_i$ may be different from $\#W_j$.

4.1 Completely Connected Network

In this subsection we provide a formula for computing in constant time, the optimal residence set in a completely connected network (i.e. a clique), denoted $G(V,E)$. We show that if $\alpha < |V|-1$, then the optimal residence set consists of one processor, and if $\alpha \geq |V|-1$, then the optimal residence set consists of all processors.

The proof for this fact is as follows. Given a residence set, R, for every processor j in R, r_j=0; for every processor k that is not in R, r_k=1. Thus, the total read cost, $\sum_{i \in V} r_i = |V| - |R|$. For every processor j in R, $w_j = |R| - 1$ and for every processor k that is not in R, $w_k = |R|$. Thus, the total write cost, $\sum_{i \in V} w_i = (|V| - |R|) \cdot |R| + |R| \cdot (|R| - 1) = (|V| - 1) \cdot |R|$. Overall,

$$cost(R) = \sum_{i \in V} w_i + \alpha \cdot \sum_{i \in V} r_i = |R| \cdot (|V| - \alpha - 1) + |V| \cdot \alpha. \qquad (1)$$

Expectedly, the actual residence set is irrelevant, only its size determines the communication cost. If $\alpha < |V| - 1$ then the minimum of the $cost(R)$ function is obtained when $|R|=1$, and if $\alpha > |V| - 1$ then the minimum is obtained when $|R| = |V|$. If $\alpha = |V| - 1$ then the costs of all possible residence sets are equal. Obviously, the size of the optimal residence set can be computed in constant time.

Unbalanced load: Let R be some residence set. Consider how the cost of R changes, when adding to it a processor, i, that does not belong to R. The total write cost, increases by $\sum_{j \in V - \{i\}} \#W_j$, since the cost of each write by all the processors, except i, increases by one. The total read cost decreases by $\#R_i$. Therefore, the total increase in cost is $\sum_{j \in V} \#W_j - (\#R_i + \#W_i)$. Consequently, the optimal residence set is:

$$\{ k \mid \sum_{j \in V} \#W_j \leq (\#R_k + \#W_k) \}$$

If there is no processor which satisfies the condition, then the optimal residence set consists of one processor, the one for which $\#R_k + \#W_k$ is maximal. The optimal residence set can obviously be found in linear (although not constant) time.

4.2 Tree Network

In this subsection we first present a simple algorithm, called TREE-RS, which determines the optimal residence set in a tree-network, $T(V,E)$. Then we prove the correctness of the algorithm, and finally, we show that the algorithm works in linear time.

In the algorithm TREE-RS we use the term median of a tree. The median is the node for which the sum of distances to the other nodes is minimal. Note that generally the median of a tree is different than its center. The latter is the node for which the *maximal* distance is minimal. Formally, in a tree let l_{vu} denote the length of the simple path between v and u. A *median* is a node, m, for which $\sum_u l_{mu}$ is minimal. It can be shown that in a tree there are one or two medians ([Z]).

The algorithm TREE-RS is given as parameters a tree network, and a read-write ratio α. Starting from some median, it incrementally constructs the optimal residence set, RS, by adding a processor to RS if it does not increase the cost of the set. The algorithm colors a processor blue if it is not in RS, and red if it is in RS, or has not been checked yet.

```
TREE-RS [ T(V,E), α]: /* Algorithm for finding the optimal residence set
                          of tree T and read-write ratio α */
1. init: color all the processors of V by red; initialize RS to a median, m.
2.     while there exists a processor in RS with at least one red neighbor, j, do:
3.         if $cost(RS \cup \{j\}) \le cost(RS)$ then add j to RS; else color j by blue.
4.     end while.
5. output: RS.
```

Next we prove that RS is an optimal residence set. The proof is concluded by Theorem 4.2.1, and uses 5 lemmas. The first states that the optimal residence set must induce a connected subgraph of the network.

Lemma 4.2.0: Assume that a residence set R induces a disconnected graph in a tree network. Then there is another residence set, R', such that $cost(R') < cost(R)$, and $|R| > |R'|$.

Proof: The construction of R' proceeds as follows. Since the graph induced by R is not connected, there must be at least two processors of R, i and j, such that if we denote the unique path between them by $i,b_1,\ldots,b_k,j$ for $k \ge 1$, then the b_i's do not belong to R. Take the pair i,j of such processors, with minimal distance between them. To obtain R' we add to R all the processors on the path between i and j. The total read cost for R' is less than the total read cost for R, since $R \subset R'$. The write cost of a processor l, for R' is equal to its write cost for R, for the following reason. There is a minimum spanning tree of $D_G(R \cup \{l\})$ in which the path between i and j is either $i—j$, or $i—l—j$. In both cases, the path can be replaced by $i,b_1,\ldots,b_k,j$ or $i,b_1,\ldots,b_h,l,b_{h+1},\ldots,b_k,j$, respectively, to obtain an equal weight minimum spanning tree of $D_G(R' \cup \{l\})$. □

Denote the processors of a graph G by $V(G)$. For the rest of this subsection, let R be a residence set that induces a connected subgraph of the tree-network, and assume that processor i 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. It disconnects the network into two subtrees: a subtree that contains i, denoted T_i, and a subtree that contains R, denoted T_R. Figure 4.2.1 illustrates the description.

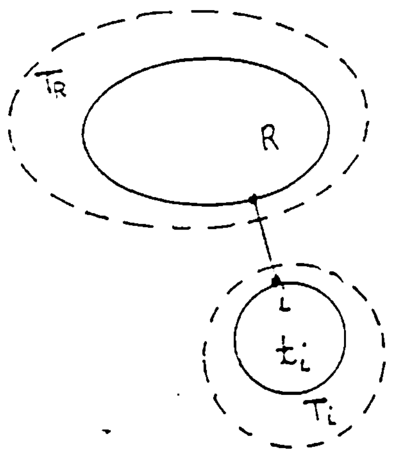


figure 4.2.1.

Lemma 4.2.1: $cost(R \cup \{i\}) = cost(R) - \alpha \cdot |V(T_i)| + |V(T_R)|$.

Proof: Observe that adding i to R decreases the read cost of each processor of T_i by one, and does not change the read cost of T_R's processors. Also, adding i to R increases the write cost of each processor of T_R by one, and does not change the write cost of T_i's processors. The lemma follows. □

Lemma 4.2.2: Let t_i be a subtree of the network such that $i \in V(t_i)$ and $V(t_i) \cap R = \emptyset$ (see figure 4.2.1). Then, $cost(R \cup \{i\}) > cost(R)$ implies that $cost(R \cup V(t_i)) > cost(R)$.

Proof: Consider how the addition of $V(t_i)$ to R changes costs. It decreases the read cost of each processor of T_i by at most $|V(t_i)|$, and does not change the read cost of T_R's processors. It also increases the write cost of each processor of T_R by $|V(t_i)|$ (by lemma 2.1), and does not decrease the write cost of T_i's processors. Note that $cost(R \cup V(t_i)) = cost(R)$ - [decrease in read cost] + [increase in write cost]. Thus, $cost(R \cup V(t_i)) \geq cost(R) - \alpha \cdot |V(T_i)| \cdot |V(t_i)| + |V(T_R)| \cdot |V(t_i)| =$ $cost(R) + |V(t_i)| \cdot (|V(T_R)| - \alpha \cdot |V(T_i)|)$. Since it is given that $cost(R \cup \{i\}) > cost(R)$, then by lemma 4.2.1, $|V(T_R)| - \alpha \cdot |V(T_i)| > 0$, and the lemma follows. □

Lemma 4.2.3: Let R' be a residence set that induces a connected subgraph, and assume that $R \subseteq R'$.

Furthermore, assume that processor i, the neighbor of some processor in R, is not in R' (see fig. 4.2.2). Then $cost(R \cup \{i\}) \leq cost(R)$ if and only if $cost(R' \cup \{i\}) \leq cost(R')$.

Proof: Straightforward from Lemma 4.2.1. □

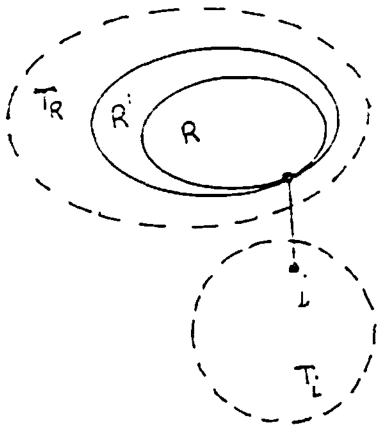


figure 4.2.2.

The last lemma indicates that for each median there is an optimal residence set which contains it. It uses the following *median property* (see [Z, KRS]). A node m is a median of a tree T, if and only if for each neighbor, w, of m it is true that: if the edge (m,w) is removed, then in the connected component which contains w there are at most half the number of nodes in T.

Lemma 4.2.4: Let m be a median. If $m \notin R$ then there is another residence set, R', such that R' induces a connected subgraph of the tree network, and $m \in R'$, and $|R| = |R'|$, and $cost(R') \leq cost(R)$.

Proof: Intuitively, the proof shows that if R is "shifted" towards m then the cost cannot increase. Formally, consider the unique path in the network from m to the closest member of R. Denote by k the last node on this path, which is not in R (see fig. 4.2.3). If R contains more than one member, then denote by j a leaf of the subtree induced by R, which is not a neighbor of k. Otherwise let j be the unique member of R. Let $R' = (R \cup \{k\}) - \{j\}$. We shall show that $cost(R') \leq cost(R)$. Denote by T_k the subtree of the network which contains k, and is obtained by removing the edge between k and its neighbor in R. Similarly, we define T_j (again, consult fig. 4.2.3). It is easy to verify that

$cost(R') = cost(R) + |V(T_j)| - |V(T_k)| + \alpha(|V(T_j)| - |V(T_k)|)$. The *median property* implies that $|V(T_k)| \geq |V(T_j)|$, and therefore $cost(R') \leq cost(R)$. The procedure can be repeated until m is contained in R'. □

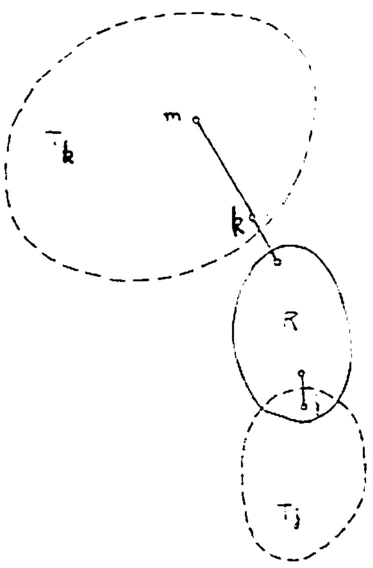


figure 4.2.3.

Theorem 4.2.1: For any tree-network and any read-write ratio, the residence set RS output by the algorithm $TREE-RS$, is optimal.

Proof : We will prove that RS satisfies the following condition. It is a minimal cost residence set that contains m, with a maximal number of processors. By lemma 4.2.4, a residence set which satisfies the condition is an optimal residence set. Assume that RS does not satisfy the condition, and denote by RS' a residence set which does so. This obviously implies that $RS \neq RS'$. The set RS' induces a connected subgraph of the network (by Lemma 4.2.0), and RS induces a connected subgraph (by the way $TREE-RS$ adds nodes to the residence set). We will analyze two cases.

case 1: $RS \subseteq RS'$. Observe that the graph induced in the network by $RS'-RS$ is a forest. Consider a tree, t_i, of this forest. It contains a processor, i, that has a neighbor in RS. Step 3 of $TREE-RS$ must have been executed for i, but it did not add i to R. In other words, adding i would have increased the cost. From Lemmas 4.2.2 and 4.2.3 we can conclude that by removing the processors of t_i from RS' a lower cost residence set can be obtained. Contradiction to the minimality of RS''s cost.

case 2: $RS \not\subseteq RS'$. Let k be the first processor that the algorithm adds, such that $k \in RS$ and $k \notin RS'$. Denote by S the residence set that the algorithm TREE-RS has, before adding k. Note that $S \neq \phi$ (because at least $m \in S$), and $S \subseteq RS'$. The algorithm adds k to S, so $cost(S \cup \{k\}) \leq cost(S)$. The processor k is a neighbor of some processor in S. By Lemma 4.2.3, $cost(RS' \cup \{k\}) \leq cost(RS')$. If $cost(RS' \cup \{k\}) < cost(RS')$ then RS' is not an optimal residence set, and if $cost(RS' \cup \{k\}) = cost(RS')$ then RS' does not have a maximal number of processors. □

Now consider the time complexity of TREE-RS. A median can be found by using the *median property*. It necessitates establishing in advance, for each edge e of the tree, how many nodes are in each connected component, if e is removed. This can be done in linear time, and then finding the median simply involves scanning the nodes. In step 2 of the algorithm every processor of the tree-network is examined at most once. For the red processors step 3 is performed. By using Lemma 4.2.1, which indicates how the addition of a processor to the residence set changes costs, step 3 can be performed in constant time. Specifically, $cost(R \cup \{j\} \leq cost(R)$ if and only if $\alpha \cdot |V(T_j)| \geq |V(T_R)|$. Therefore, the optimal residence set can be found in linear time.

Unbalanced load: The algorithm TREE-RS is still performed as given, to obtain the optimal residence set, except that the set RS, in step 1, is initialized to a *weighted median*. It is the node, m, for which $\sum_u (\#R_u + \#W_u) \cdot l_{mu}$ is minimal. A weighted median can still be found in linear time using the following *weighted median property*. A node is a weighted median if and only if for each neighbor, w, of m it is true that: if the edge (m,w) is removed, then

$$\sum_{u \in the-connected-component-that-contains-m} (\#R_u + \#W_u) \geq \sum_{u \in the-connected-component-that-contains-w} (\#R_u + \#W_u).$$

The proof that RS is an optimal residence set can be repeated almost verbatim, except that Lemma 4.2.1, that is also used in step 3 of TREE-RS, becomes: $cost(R \cup \{i\}) = cost(R) - \sum_{j \in V(T_i)} \#R_j + \sum_{j \in V(T_R)} \#W_j$.

4.3 Ring

The main result of this subsection is providing in theorem 4.3.1 a formula to compute the optimal residence set in constant time. First we prove 4 lemmas, enabling us to conclude that for any size ring, and for any read-write ratio, there is an optimal residence set that induces a connected subgraph of a ring network. We refer to a connected subgraph of the ring network as a *string*. The first lemma enables us to speak subsequently in more intuitive terms of strings, rather than the distance-graph.

Lemma 4.3.1: Let G be a ring network, R be a residence set, and i be a processor of G. Then w_i, the write cost of processor i, equals to the number of edges in the shortest string of G that contains all the processors of $R \cup \{i\}$.

Proof : Obvious.

Given a residence set, R, in a ring network, consider the subgraph, Q, induced by the processors of the network that do not belong to R. A connected component of Q is a *hole*. For example, in fig. 4.3.1a, H_s and H_b are holes.

Lemma 4.3.2: For any residence set R which induces three or more strings in a ring network, there is a residence set having a lower cost.

Proof : Denote by H the set of processors, each of which is not in R, and is not in the two biggest holes of R. Since there are more than two holes, H is not empty. Consider the residence set, R', which is $R \cup H$ (we fill all but the two biggest holes). Since R' is a proper superset of R, the total cost of reads for R' is less than for R. Consider an arbitrary processor, v. If v does not belong to in the biggest hole of R, then it is easy to see, by Lemma 4.3.1, that its write cost is:

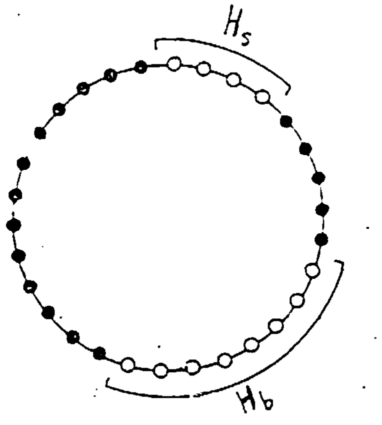
(number of edges in the network) – (number of edges in the the biggest hole) – 2

regardless of whether R or R' is the residence set.

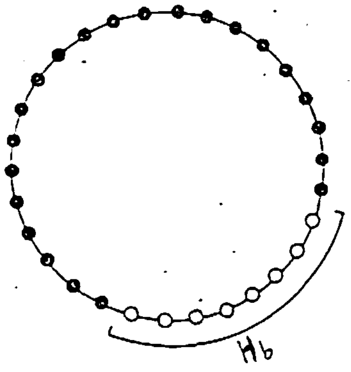
If v belongs to the biggest hole of R, then, given the residence set R', there are two possibilities: either v writes by "skipping" part of the biggest hole (fig. 4.3.2a), or it writes by "skipping" the whole second biggest hole (fig. 4.3.2b). However, notice that in both cases, if R is the residence set, then v writes in the same fashion. Therefore, the total write cost of all the processors given the residence set R, is equal to the total write cost of all the processors given the residence set R'. Thus, since the total cost for reads is smaller for R', $cost(R') < cost(R)$. □

Lemma 4.3.3: If the read-write ratio $\alpha > 1$, then for any residence set that induces two strings, there is a residence set that has a lower cost.

Proof : Let R be a residence set that induces two strings (see fig. 4.3.1a). Denote by H_s and H_b the sets of processors of the smaller hole and of the bigger hole, respectively (of course, their size may be equal). Consider the residence set, R', which is $R \cup H_s$ (see fig. 4.3.1b). We will show that $cost(R') < cost(R)$.



a



b

figure 4.3.1: A ring. Shaded processors are members of the residence set.

For each processor in $R \cup H_b$, the read costs using R and R' are equal. The read cost of a processor of H_s is zero using R', and nonzero using R. First we compute the total decrease in the cost of reads, when substituting R' for R.

case 1: $|H_s|$ is even.

Then there are two processors in H_s at distance i from R, for each $1 \le i \le \frac{|H_s|}{2}$. The total read cost decreases by $2 \cdot \sum_{i=1}^{\frac{|H_s|}{2}} i = \frac{|H_s|}{2} \cdot (\frac{|H_s|}{2} + 1)$. (3)

case 2: $|H_s|$ is odd.

Then there are two in H_s processors at distance i from R, for each $1 \le i \le \frac{|H_s|-1}{2}$, and there is one processor at distance $\frac{|H_s|+1}{2}$. The total read cost when using R' instead of R decreases by $2 \cdot \sum_{i=1}^{\frac{|H_s|-1}{2}} i + \frac{|H_s|+1}{2} =$

$$(\frac{|H_s|+1}{2})^2. \qquad (4)$$

Now consider the write costs. The write cost of a processor in $R \cup H_s$ has not changed, because by Lemma 4.3.1 it equals the number of edges in the string induced by $R \cup H_s$; and this is the same string (i.e. the ring minus the biggest hole) for R or R'. In H_b however, there are two types of processors (see fig. 4.3.2).

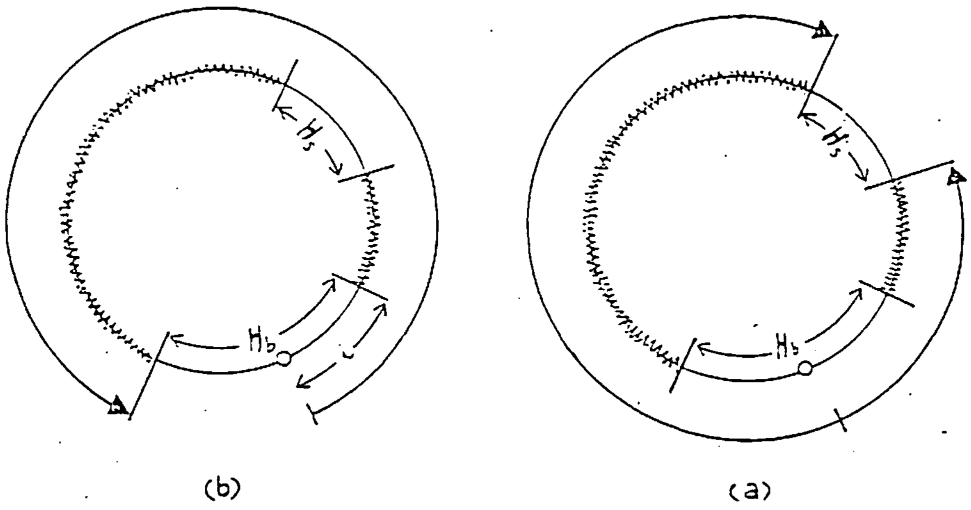


figure 4.3.2: H_b is the biggest hole, and H_s is of equal, or smaller, size.

The first type (fig. 4.3.2a) are processors for which a shortest string that contains them and R, contains the whole ring except some processors of H_b. For these processors the shortest string has not changed, and by lemma 4.3.1 their write cost has not changed. Therefore, if H_b contains only processors of the first type, then the total cost is decreased for R' compared to R, and the proof is completed.

The second type of processors in H_b (fig 4.3.2b) are processors for which the shortest string contains the whole ring, except the processors of H_s. Precisely, let i be the distance from R of a processor in H_b. Then, the processor is of *type two* if $|H_s| > |H_b| - i$, or in other words, $i > |H_b| - |H_s|$; otherwise it is of *type one*. For a processor of type two, the shortest string that contains it and the processors of R' is longer than the shortest string that contains it and the processors of R. Denote by Δl the increase in the total write cost, when substituting R' for R. The following two cases give an upper bound on Δl. From this upper bound, the fact that $\alpha > 1$, and formulas (3) and (4), the lemma follows.

case 1: $|H_b|$ is even.

Then there are two processors at distance i for each $|H_b|-|H_s|<i\leq\frac{|H_b|}{2}$, and $\Delta l=2\cdot\sum_{i=|H_b|-|H_s|+1}^{\frac{|H_b|}{2}}(|H_s|+i-|H_b|)=(1+|H_s|-\frac{|H_b|}{2})\cdot(|H_s|-\frac{|H_b|}{2})$. Given that $|H_b|\geq|H_s|>\frac{|H_b|}{2}$, if $|H_s|$ is even then $\Delta l\leq\frac{|H_s|}{2}\cdot(1+\frac{|H_s|}{2})$, and if $|H_s|$ is odd then $\Delta l\leq(\frac{|H_s|+1}{2})^2$.

case 2: $|H_b|$ is odd.

Then there are two processors at distance i, for each $|H_b|-|H_s|<i\leq\frac{|H_b|-1}{2}$, and one processor at distance $\frac{|H_b|+1}{2}$. It can be easily verified that $\Delta l=(2\cdot\sum_{i=|H_b|-|H_s|+1}^{\frac{|H_b|-1}{2}}(|H_s|+i-|H_b|))+(|H_s|-\frac{|H_b|-1}{2})=(|H_s|-\frac{|H_b|-1}{2})^2$. Again, given that $|H_b|\geq|H_s|$, if $|H_s|$ is even then $\Delta l\leq\frac{|H_s|}{2}\cdot(1+\frac{|H_s|}{2})$, and if $|H_s|$ is odd then $\Delta l\leq(\frac{|H_s|+1}{2})^2$. □

Lemma 4.3.4: Let the read-write ratio $\alpha\leq 1$, and R be a residence set which induces two strings, and denote the smallest hole by H_s. Then there is another residence set R' that induces at most two holes, and $cost(R')\leq cost(R)$, and $|R'|=|R|$, and the smallest hole it induces, H'_s, is of a smaller size than H_s.

Proof: We shall define R' to be the "shift" of the long string induced by R, closer by one position to the short string (thus reducing $|H_s|$ by one). Denote by S_b, S_s the big and small strings induced by R, respectively. Denote by n_s the processor of H_s which is closest to S_b, and by n_b the processor of S_b which is closest to H_b (see Fig. 4.3.3).

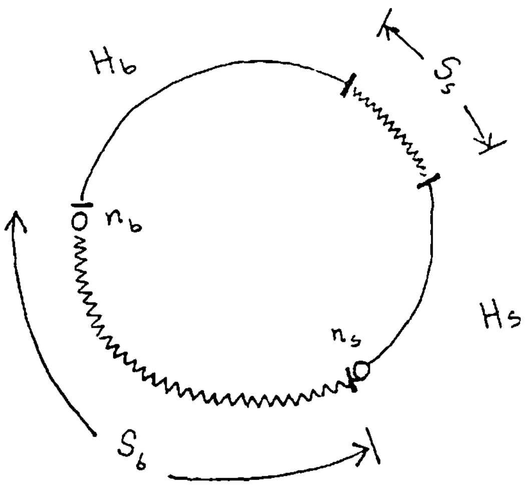


figure 4.3.3.

Let $R' = S_s \cup (S_b - \{n_b\}) \cup \{n_s\}$. It is easy to calculate that the total read-cost is higher for R' than for R, by

$$\alpha\cdot(\left\lfloor \frac{|H_b|}{2} \right\rfloor - \left\lceil \frac{|H_s|}{2} \right\rceil + 1).$$

Consider now the write costs. We shall show that there is a decrease in the total write cost, and this decrease is at least as high as the increase in read costs. Since $|H'_b| = |H_b| + 1$ the write cost of a processor of $S_s \cup H_s \cup \{S_b\} - \{n_b\})$ is lower by one for R' than for R. For n_b, the write-cost is equal for the two residence sets. Left is only to evaluate the change in write costs for the processors of H_b. Assume that the processors of H_b do not change type (see proof of lemma 4.3.3 for the definition of type) when going from R to R'. In other words, if a processor of H_b writes R by skipping part of H_b (fig. 4.3.2a), then it writes R' by skipping part of H'_b, and if it writes R by skipping H_s, then it writes R' by skipping H'_s. If all the processors of H_b are of type one, then the write cost of $\left\lceil \frac{|H_b|}{2} \right\rceil$ processors is lower by one for R' than for R.

Overall, the total write cost is lower for R' than for R by $|S_s| + |H_s| + |S_b| + \left\lceil \frac{|H_b|}{2} \right\rceil - 1$; since $\alpha \leq 1$

$cost(R') - cost(R) \leq \left\lceil \frac{|H_b|}{2} \right\rceil - \left\lceil \frac{|H_s|}{2} \right\rceil + 1 - |S_s| - |H_s| - |S_b| + 1 - \left\lceil \frac{|H_b|}{2} \right\rceil < 0$, and the lemma follows.

Suppose now that there are processors of type two in H_b (i.e. processors for which their distance from R is more than $|H_b| - |H_s|$). There are $|H_b| - 2\cdot|H_s|$ such processors. Then the write cost of the $|H_b| - |H_s|$ processors of type one is lower by one for R' than for R; the write cost of the $|H_b| - 2\cdot|H_s|$ processors of type two is higher by one for R'. Then the total write cost is lower for R' than for R by

$$|S_s|+|H_s|+|S_b|-1+|H_b|-|H_s|-2\cdot|H_s|+|H_b| = |S_s|+|S_b|+2\cdot|H_b|-1-2\cdot|H_s|.$$

Overall, since $\alpha \le 1$,

$$cost(R')-cost(R) \le \left\lfloor \frac{|H_b|}{2} \right\rfloor - \left\lceil \frac{|H_s|}{2} \right\rceil + 1 - |S_s| - |S_b| - 2\cdot|H_b| - 1 + 2\cdot|H_s| \le 0.$$

We have assumed that the processors of H_b do not change type when going from R to R'. If they do, it is because their write cost can be decreased further, and our inequalities obviously continue to hold. □

Corollary 4.3.1: If the read-write ratio $\alpha \le 1$, then for any residence set R that induces two strings, there is another residence set R' that induces only one string, and $cost(R') \le cost(R)$, and $|R| \le |R'|$.

Proof : Immediate from Lemma 4.3.4, since the size of the smaller hole can iteratively be reduced to zero. □

By Lemma 4.3.3 and Corollary 4.3.1, we conclude that for each ring, and for each read-write ratio, there exists an optimal residence set with at most one hole; in other words, there exists an optimal residence-set that induces a connected subgraph of the network. Because of the ring topology, and the fact that all the processors have the same access pattern, it is clear that all the residence sets that induce a string of the same size, have the same cost. Therefore, we only have to provide a formula for computing the cardinality of the optimal residence set. This we do in the next theorem.

Theorem 4.3.1: Let n be the size of the ring, and α be the read-write ratio. Then the cardinality of the optimal residence set equals to $\frac{n\cdot(\alpha-1)}{\alpha+1}+1$.

Proof : If R is a string of size k, then for $v \in R$ the cost $w_v = k-1$ and the cost $r_v = 0$; for $v \notin R$ at distance i from R, $w_v = i+k-1$ and $r_v = i$. Summing up we obtain: $cost(R) = n\cdot(k-1) + \frac{\alpha+1}{4}\cdot[(n-k+1)^2-c]$, where $c=1$ if $n-k$ is even, and $c=0$ if $n-k$ is odd. In both cases, the derivative of $cost(R)$ with respect to k is $n + \frac{1}{2}\cdot(\alpha+1)(k-n-1)$. This indicates that the minimum of $cost(R)$ is obtained when $k = \frac{n(\alpha-1)}{\alpha+1} + 1$. □

Unbalanced Load: Only lemmas 4.3.1 and 4.3.2 still hold in case each processor has a different access pattern ($\#R_i$ and $\#W_i$). However, lemma 4.3.2 is enough to provide a polynomial (although not constant) time algorithm. Specifically, the fact that we know that in an optimal residence set there are at most two holes, enables finding the optimal residence set as follows. Assume that the ring is of size n. It is easy to see that there are $O(n^4)$ different residence sets, each with at most two holes. Finding the cost of a residence set can be done in $O(n)$. Consequently, the time complexity of finding the optimal residence set is $O(n^5)$.

5. Reliability Considerations

As mentioned in the introduction, often the number of processors in the residence set should not fall below some threshold, t. Therefore, although the optimal residence set contains, for example, one processor, this may be unacceptable for reliability reasons. Consequently, we are often interested in finding the t-reliable optimal residence set. The resulting $t-RS$ problem is formally defined as follows, for the balanced-load case (unbalanced load is addressed at the end of this section). Given a network, $G = (V,E)$, and a read-write ration, α, and a reliability threshold, $t < |V|$, what is the residence set $R \subseteq V$ such that $|R| \geq t$ and $cost(R)$ is minimal? The problem remains NP-complete (we have proved it NP-complete for $t=1$) but for the special topologies discussed in Section 4, it can be solved efficiently.

Consider first a completely connected network. If the cardinality of the optimal residence set is less than the threshold t, it means that $\alpha < |V|-1$, and that the optimal residence set is of size one. But then, based on formula (1) in subsection 4.1, it is easy to see that the solution to the $t-RS$ problem is any subset of t processors.

Next, consider a ring network. Based on Lemma 4.3.3 and on Corollary 4.3.1, we can disregard residence sets which induce more than one connected component. Then the formula developed in the proof of Theorem 4.3.1 for $cost(R)$ as a function of the size of the residence set, is a polynomial of rank two having a minimum value. Consequently, since t is bigger than the optimal residence set size (otherwise the problem is trivial), the solution to the $t-RS$ problem is a string of size t.

Finally, we consider a tree network. For this network topology the solution is slightly more complicated. We present a quadratic time algorithm, TREE-tRS, which provides a solution to the t-RS problem. Given a tree T, a residence set R which induces a connected subgraph of T, and a node j of T which is not in R, but is a neighbor of some node i in R assume that we remove the edge (i,j) from T; then remember that we denote by T_j the connected component which includes j, and by T_R the other one. The algorithm is formally given below, but intuitively, it starts with RS, the optimal residence set output by TREE-RS, and iteratively adds to it the neighbor j for which $|T_j|$ is minimal. Obviously, each addition of a node to RS increases the communication cost.

```
$TREE-tRS[T(V,E),\alpha,t]$;
1. initialize $R_t$ to $RS$, the optimal residence set output by $TREE-RS[T(V,E),\alpha]$;
2. while $|R_t| < t$ do;
3.      add to $R_t$ the neighbor $j$ for which $|T_j|$ is maximal;
4. end;
5. output $R_t$.
```

Next we shall prove that the set R_t output by TREE-tRS indeed provides a solution to the t-RS problem. First we need some notation. Denote by RS_t a solution to the $t-RS$ problem which includes the median, m, used by TREE-RS (according to Lemma 4.2.4 there is such), and is of maximal size. Obviously, $|RS_t| \geq t$. By Lemma 4.2.0, RS_t induces a connected subtree of the network. Denote by RS the optimal residence set obtained by the procedure $TREE-RS$, when starting from m. We obviously suppose that $|RS| < t$, otherwise RS is a solution to the $t-RS$ problem.

Lemma 5.1: $RS \subset RS_t$.

Proof: Suppose that $RS \not\subset RS_t$. Note that $RS \cap RS_t \neq \emptyset$, because m belongs to both sets. Therefore, consider a processor $i \in RS-RS_t$ which is a neighbor of a processor in RS_t (see Fig. 5.1). By Lemma 4.2.3, and the fact that $TREE-RS$ added i to the residence set, it is clear that $cost((RS \cap RS_t) \cup \{i\}) \leq cost(RS \cap RS_t)$. Then, again by Lemma 4.2.3, $cost(RS_t \cup \{i\}) \leq cost(RS_t)$. But if $cost(RS_t \cup \{i\}) < cost(RS_t)$ it contradicts the cost minimality of RS_t, and if $cost(RS_t \cup \{i\}) = cost(RS_t)$ it contradicts the maximality of the size of RS_t. □

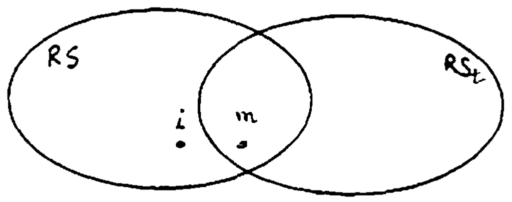


figure 5.1

Theorem 5.1: $cost(RS_t) = cost(R_t)$.

Proof: First we shall show that in RS_t there are exactly t processors. By Lemma 5.1 and step 3 of algorithm TREE-RS, the cost of RS_t is strictly higher than $cost(RS)$ (remember also that we assumed that $|RS| < t$). Thus, if $|RS_t| > t$, then, by Lemma 4.2.1, at least one processor can be omitted from RS_t, to obtain a residence set of lower cost, that is of size $\geq t$. This contradicts the cost minimality of RS_t.

Next we shall show that RS_t can be transformed into R_t by a sequence of add-processor-drop-processor transformations, such that each transformation is cost preserving. Denote $H = R_t \cap RS_t$ (see fig. 5.2).

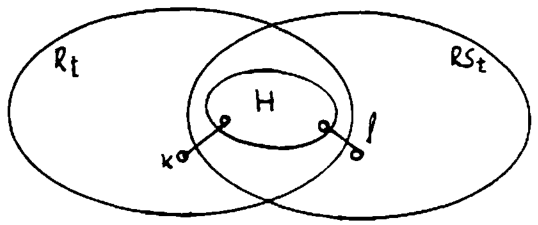


figure 5.2

By Lemma 5.1 $H \supseteq RS$, and consequently, $H \neq \emptyset$. If $RS_t - H = \emptyset$, or $R_t - H = \emptyset$, then $RS_t = R_t$, since both sets are of the same size. Then the theorem trivially follows. Assume that $RS_t - H \neq \emptyset$, and consequently $R_t - H \neq \emptyset$. We shall show that every processor in $RS_t - H$ is a neighbor of H, and so is every processor in $R_t - H$; furthermore, the add-processor-drop-processor cost preserving transformations, consist of dropping any node in $RS_t - H$, and adding any node in $R_t - H$. Since $RS_t - H$ and $R_t - H$ are of the same size, the theorem follows.

Let l be some leaf of the subtree induced by RS_t, such that l is not in H (see fig. 5.2). Let k be some processor in $R_t - RS_t$ which is a neighbor of H (see Fig. 5.2). By Lemma 4.2.1, $cost(RS_t - \{l\}) = \alpha \cdot |V(T_l)| - |V(T_{RS_t - \{l\}})| + cost(RS_t)$. If l is not a neighbor of H, then by step 3 of $TREE{-}tRS$, $|V(T_l)| < |V(T_k)|$; therefore, $cost((RS_t - \{l\}) \cup \{k\}) < cost(RS_t)$, and this contradicts the cost minimality of RS_t. Consequently, l is a neighbor of H. l is an arbitrary leaf of the subtree induced by RS_t, that is not in H. Therefore any such leaf is a neighbor of H, and consequently, any internal processor of the subtree is in H. Thus any processor of $RS_t - H$ is a neighbor of H. Now let l denote an arbitrary processor of $RS_t - H$. It is easy to see, again by Lemma 4.2.1, that adding k to $RS_t - \{l\}$ gives a residence set, Q, whose cost is not higher than the cost of RS_t; otherwise step 3 of $TREE{-}tRS$ would have added l to R_t before adding k. The cost of Q cannot be lower than the cost of RS_t, because of the way RS_t was defined. In other words, by Lemma 4.2.1, T_l has exactly as many nodes as T_k. Suppose that $R_t - H$ contains a processor, g, that is not a neighbor of H; then in step 3 of $TREE-tRS$, l would have been selected before the selection of g, since l has as many nodes as k. Thus any processor g of R_t-H is a neighbor of H, and T_g has exactly as many nodes as T_k. Therefore, any processor of RS_t-H can be replaced in RS_t by any processor of R_t-H, to obtain a residence set of cost equal to $cost(RS_t)$. □

Unbalanced load.

Completely connected network: This case can be resolved based on the case unconstrained by reliability considerations (subsection 4.1). The optimal residence set, RS, found there for the unbalanced-load case, should be extended by adding to it $t-|RS|$ processors, in the following fashion. Sort the processors that are not in RS, in decreasing order of their sum $\#R_i + \#W_i$. Then add to RS the first $t-|RS|$ processors in the sequence so obtained. As explained in subsection 4.1, this will increase the cost of RS the least.

Tree network: The algorithm TREE-tRS can be extended to provide the t-reliable optimal residence set, in an unbalanced-load environment. Assume that processor i in the network performs $\#R_i$ reads per time-unit, and $\#W_i$ writes per time unit. Then the algorithm TREE-tRS can still be executed as given, except that step 3 becomes:

add to R_t a neighbor j for which $\sum_{u-is-a-processor-of-T_j} \#R_u - \sum_{u-is-a-processor-of-T_R} \#W_u$ is maximal.

Lemma 5.1, and Theorem 5.1 continue to hold, and their proof is identical to the one given for the balanced-load case.

Ring network: This case can be handled in the same fashion, as the case unconstrained by reliability considerations (subsection 4.3). The cost of each residence set of size at least t, and having at most two holes, is be computed. The minimal-cost such set is the desired one.

6. Discussion

In this paper we first proposed a new method by which a processor should write all replicas of a data item for minimizing communication. The processor should construct a minimum spanning tree of what we called the distance graph, and then propagate the data item along its edges. The read, as usual, is carried out from the closest replica. Then we showed that determining a residence set for minimizing overall communication is *NP−Complete* in networks modeled by general graphs. However, we provided constant time algorithms for determining the optimal residence set in completely-connected networks and rings, and a linear time algorithm for determining the optimal residence in tree-networks. Extensions for the algorithms in case reliability constraints exist, were provided.

Next, we would like to demonstrate that the communication cost improvement obtained by using the residence sets proposed in this paper is significant. Consider, for example, in a ring network of n processors, how the proposed residence set compares with a trivial residence set consisting of all n processors. Denote the number of messages per time unit for this trivial residence set by $cost_n$, and the number of messages per time unit for the optimal residence set (Theorem 4.3.1) by $cost_{opt}$. Then, if the read-write ratio is α, $cost_{opt} = \frac{n^2\alpha}{\alpha+1}$, and $cost_n = n(n-1)$. Furthermore, $\frac{cost_n}{cost_{opt}} \xrightarrow[n\to\infty]{} 1+\frac{1}{\alpha}$. Therefore, if for example $\alpha = 2$, then as the number of processors grows, our proposed residence set is 33% better than the trivial one. Similarly, one can show that if we consider another trivial residence set, consisting of one processor, then $\frac{cost_1}{cost_{opt}} \xrightarrow[n\to\infty]{} \frac{(\alpha+1)^2}{4\alpha}$, and for $\alpha = 4$, a 36% gain is realized.

An additional remark is that although the optimal set for the discussed topologies induces a connected graph, this is not necessarily the case in a general network. For example, for the network in fig 6.1 and for α=1.8 the unique optimal residence set is {4,8}.

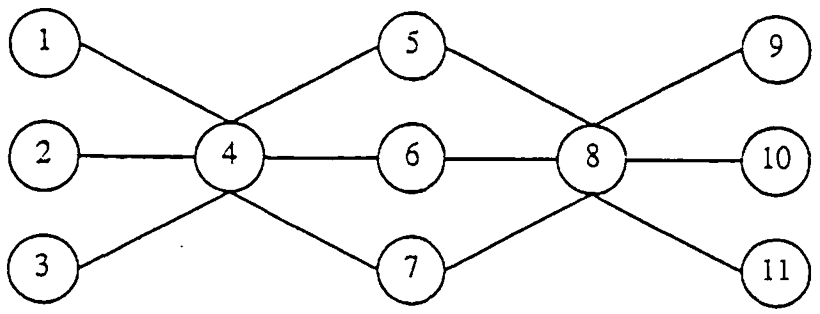


figure 6.1.

As far as future work is concerned, much remains to be done. First, it would be interesting to generalize our results to other network topologies, e.g. the hypercube. Second, bounded error approximations for general networks should be investigated. Third, what happens when communication time in addition to communication cost is an issue (such an analysis was carried out in [SW])? Finally, it would be interesting to generalize the results to majority-voting access schemes, as opposed to read-one-write-all.

Acknowledgement: We wish to thank the referees for helping us improve the focus and presentation of this paper. One of the referees also showed us how to solve the optimal residence set problem for completely connected networks, in the unbalanced-load case. This prompted our extension of all the results, to handle the unbalanced-load case.

References

[C] R. G. Casey, "Allocation of Copies of a File in an Information Network", Proc. 1972 Spring Joint Computer Conference, AFIPS, 1972.

[CD] D. Cheriton and S. E. Deering "Hosts Groups: A Multicast Extension for Datagram Internetworks", Stanford University Manuscript.

[CZ] D. Cheriton and W. Zwaenepoel "Distributed Process Groups in the V Kernel", ACM Transactions on Computer Systems, 3 :2, 1985.

[CMP] S. Ceri, G. Martella, and G. Pelagatti, "Optimal File Allocation in a Computer Network: a Solution Method Based on the Knapsack Problem", Computer Networks, **6** :5, 1982.

[D] Y. K. Dalal, "Broadcast Protocols in Packet Switched Computer Networks", Ph.D. Thesis,, Stanford University, DSL TR 128, 1977.

[De] S. E. Deering "Multicast Routing in Internetworks, and Extended LANs", Stanford University Manuscript.

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

[DM] Y. K. Dalal and R. M. Metcalfe "Reverse Path Forwarding of Broadcast Packets", Communications of the ACM, **21** :12, 1978.

[FH] M. L. Fisher and D. S. Hochbaum, "Database Location in Computer Networks", Journal of the ACM, **27** :4, 1980.

[IK] K. B. Irani and N. G. Khabbaz, "A Methodology for the Design of Communication Networks and the Distribution of Data in Distributed Supercomputer Systems", IEEE Transactions on Computers, **C-31** :5, 1982.

[KRS] E. Korach, D. Rotem, and N. Santoro, "Distributed Algorithms for Finding Centers and Medians in Networks", ACM Transactions on Programming Languages and Systems, **6** :3, 1984.

[M] A. Milo, M.Sc. Thesis, The Technion - IIT, Jan. 1988 (in hebrew).

[ML] H. L. Morgan and J. D. Levin "Optimal Program and Data Location in Computer Networks", Communications of the ACM, **20** :5, 1977.

[MIMH] S. Muro, T. Ibaraki, H. Miyajima, and T. Hasegawa, "Evaluation of the File Redundancy in Distributed Database Systems" IEEE Transactions on Software Engineering, **SE-11** :2, 1985.

[MR] S. Mahmoud and J. S. Riordon, "Optimal Allocation of Resources in Distributed Information Network", ACM Transactions on Database Systems, **1** :1, 1976.

[P] R. Perlman, "An Algorithm for Distributed Computation of a Spanning Tree in an extended LAN", Proc. 9th Data Communications Symposium, ACM/IEEE, 1975.

[SW] A. Segall and O. Wolfson, "Transaction Commitment at Minimal Communication Cost", Proc. 6th ACM Symp on Principles of Database Systems, 1987.

[RW] C. V. Ramamoorthy and B. W. Wah, "The Isomorphism of Simple File Allocation", IEEE Transactions on Computers, **C-32** :3, 1983.

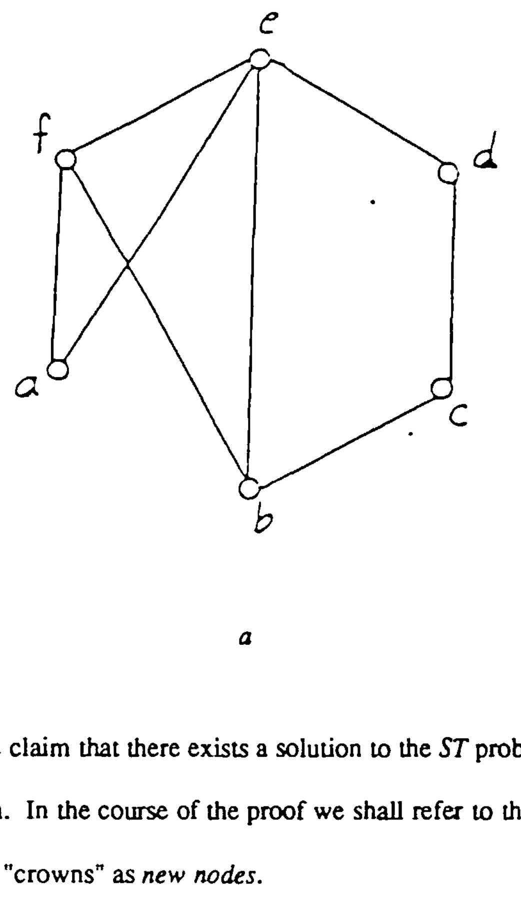
[Z] B. Zelinka, "Medians and Peripherians on Trees", Arch. Math. (Brno), 1968.

APPENDIX

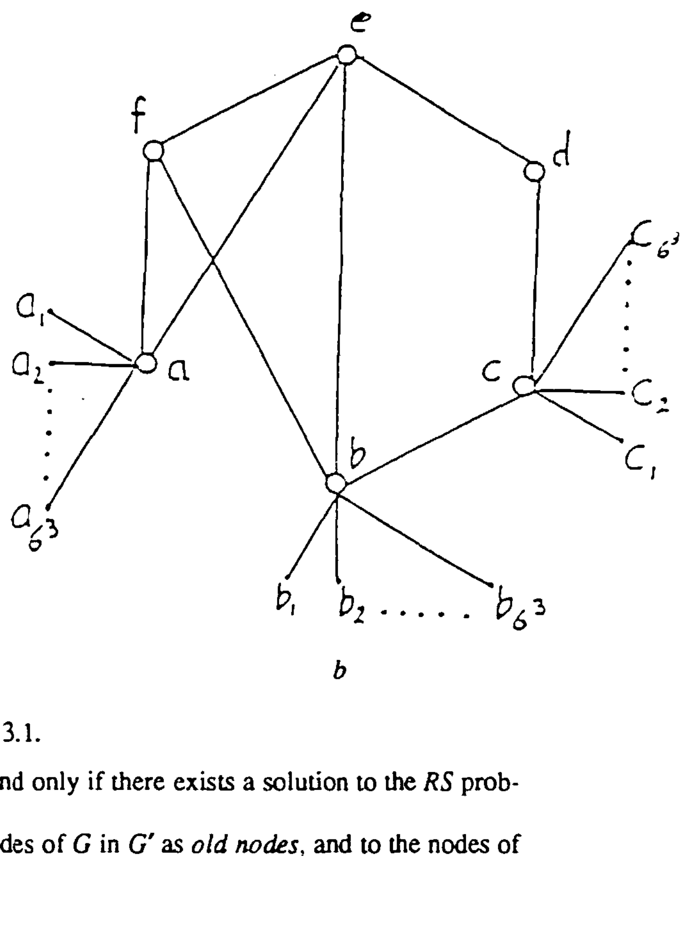
Proof of Theorem 3.1 It is easy to see that $RS \in NP$. Guess a subset $R \subseteq V$, find w_i and r_i for each $i \in V$, and verify that $\sum_{i \in V} w_i + \alpha \cdot \sum_{i \in V} r_i < C$. Obviously, this can be done in polynomial time.

Next we show that RS is NP*-Hard*. This is done by transforming the Steiner Tree (ST) problem to RS. In ST the input consists of a graph $G'=(V',E')$, a subset $X \subseteq V'$ and a positive integer $B < |V'|$. The question is whether there exists a subtree of G' that includes all the nodes of X, and such that its number of edges is no more than B. We shall assume without loss of generality that $1 < |X| < B < |V'|$.

Given an instance of the ST problem we construct an instance of the RS problem as follows. The graph G consists of G', with every node $u \in X$ connected to a "crown" of $|V'|^3$ new nodes: $u_1, u_2, \ldots, u_{|V'|^3}$. For example, in fig. 3.1b there is graph constructed from the graph of fig. 3.1a, where $X=\{a,b,c\}$. Let $\alpha = B$ and $C = 2 \cdot |V'|^3 \cdot |X| \cdot (B+1)$.



a



b

figure 3.1.

We claim that there exists a solution to the ST problem if and only if there exists a solution to the RS problem. In the course of the proof we shall refer to the the nodes of G in G' as *old nodes*, and to the nodes of the "crowns" as *new nodes*.

(only if) Assume that there exists a Steiner tree, T, in G' whose weight is no more than B. We choose as the residence set R all the nodes of T, and show that it constitutes a solution to the RS problem. Note that for every new node, say k, $r_k=1$ and for every old node, j, $r_j \le |V'|$. Therefore, the total cost for reading is $\sum_{i \in V} r_i \le |V'|^3 \cdot |X| + |V'|^2$. By lemma 2.1, for every new node, k, $w_k \le B+1$. The distance of every old node from R is at most $|V'|-B-1$, and $|R| \le B+1$, so $w_j \le |V'|$. Therefore, the total cost for writing is $\sum_{i \in V} w_i \le |V'|^3 \cdot |X| \cdot (B+1) + |V'|^2$.

Totaling, the cost is: $\sum_{i \in V} w_i + \alpha \cdot \sum_{i \in V} r_i \le |V'|^3 \cdot |X| \cdot (B+1) + |V'|^2 + B \cdot (|V'|^3 \cdot |X| + |V'|^2) \le^{(1)}$

$|V'|^3 \cdot |X| \cdot (2 \cdot B+1) + |V'|^3 <^{(2)} 2 \cdot |V'|^3 \cdot |X| \cdot (B+1) = C.$

(1) $|V'| > B$.

(2) $|X| > 1$.

Thus, the residence set R is a solution to the RS problem.

(if) Assume now that the set R constitutes a solution to the RS problem. We use 3 lemmas for this direction of the proof.

Lemma 3.0: There exists a residence set, R', which does not contain any new nodes, and $cost(R') \le cost(R)$.

Proof: To obtain R' we will repeatedly perform one of the following two transformations, for each new node $u \in R$.

Case 1: The old node v, which neighbors u, belongs to R. Then drop u from R. The read cost of every processor, except u, remains the same, and the read cost of u increases by one. On the other hand, the write cost of every node of G, except u, decreases by one, and the write cost of u remains the same. Since $\alpha < |V|-1$, the cost of the new residence set is lower than $cost(R)$.

Case 2: The old node v, which neighbors u, does not belong to R. Then drop u from R and add v to R. The read cost of any node, except u and v, obviously does not increase due to the transformation, and the read cost of u increases by one, and the read cost of v decreases by one. Overall, the total read cost does not increase. The total write cost also does not increase due to the transformation, because every path from a node to u goes through v. □

To simplify notation we shall assume that R does not contain any new nodes.

Lemma 3.1: If R does not contain X, then there exists another residence set, R', for which $cost(R') \le cost(R)$, and $X \subseteq R'$, and R' does not contain any new nodes.

Proof : Denote the nodes in X-R by {1,2,...,k}. To obtain R', first we choose for every node $h \in X-R$ a shortest path, p_h, from h to a member of R. Then we add to R all the nodes of all p_h's, and denote this new residence set by R'. Obviously, $X \subseteq R'$ and R' does not contain any new nodes. We denote by d_h the length (in arcs) of p_h for $h=1,2,...,k$. Let $d=d_1+d_2+...+d_k$. The total-read-cost for R' decreases by at least B times $|V'|^3 \cdot d_1 + |V'|^3 \cdot d_2 + ... + |V'|^3 \cdot d_k = |V'|^3 \cdot d$ (cost of reads by new nodes which are neighbors of X-R), compared to the total-read-cost for R. We add no more than d nodes to R to create R'. Note that for each node i, w_i increases by at most d. Since there are $|V'| + |V'|^3 \cdot |X|$ nodes in the graph, the total write cost of all the nodes increases by at most $d \cdot (|V'| + |V'|^3 \cdot |X|)$.

Totaling, $cost(R') \le cost(R) + (|V'| + |V'|^3 \cdot |X|) \cdot d - B \cdot |V'|^3 \cdot d =$

$cost(R) + |V'|^3 \cdot d \cdot (|X| - B) + |V'| \cdot d \le cost(R) + |V'|^3 \cdot d \cdot (|X| - B + 1) \le^{(1)} cost(R)$

(1) $B > |X|$. □

For the next lemma we need the following definition. Given an indirected graph, G, and a subset of the nodes, X, the *minimal Steiner tree* is a subgraph of G which: 1) is a tree, and 2) contains the nodes in X, and 3) has a minimal number of edges among all subgraphs which satisfy the first two conditions.

Lemma 3.2: Let G be a communication network graph, and X a subset of its nodes. The weight of a minimal spanning tree (mst) of the distance graph on X is not smaller than the weight of the minimal Steiner tree for X.

Proof : Obvious.

Assume that there is no Steiner tree with B or less edges. Lemma 3.2 implies that the total weight of $mst(D_G(X))$ is at least $B+1$. Thus, the cost of a write of a new node for R' (Lemma 3.2), is at least $B+2$ (one hop to the closest node in X, and then at least $B+1$ hops to all the nodes of X through the mst). Therefore the total write cost for R' is at least $(B+2) \cdot |X| \cdot |V'|^3$. The total read cost for R' is at least the read cost of the new nodes, i.e., $|X| \cdot |V'|^3$. Totaling,

$cost(R) \ge cost(R') = \sum_{i \in V} w_i + \alpha \cdot \sum_{i \in V} r_i \ge (B+2) \cdot |X| \cdot |V'|^3 + B \cdot |X| \cdot |V'|^3 = 2 \cdot |V'|^3 \cdot |X| \cdot (B+1) =$

But this contradicts the fact that R is a solution to the RS problem. □