Dartmouth College
Dartmouth Digital Commons

Computer Science Technical Reports
Computer Science

1-1-1986

An Algorithm for Resource Allocation Requiring Low Overhead Communication

Ann Marks
Dartmouth College

Follow this and additional works at: https://digitalcommons.dartmouth.edu/cs_tr

Part of the Computer Sciences Commons, and the Engineering Commons

Dartmouth Digital Commons Citation
Marks, Ann, "An Algorithm for Resource Allocation Requiring Low Overhead Communication" (1986). Computer Science Technical Report PCS-TR86-129. https://digitalcommons.dartmouth.edu/cs_tr/28

This Technical Report is brought to you for free and open access by the Computer Science at Dartmouth Digital Commons. It has been accepted for inclusion in Computer Science Technical Reports by an authorized administrator of Dartmouth Digital Commons. For more information, please contact dartmouthdigitalcommons@groups.dartmouth.edu.

AN ALGORITHM FOR RESOURCE ALLOCATION REQUIRING LOW OVERHEAD COMMUNICATION

Ann Marks

Technical Report PCS-TR86-129

An Algorithm for Resource Allocation Requiring Low Overhead Communication

Ann Marks

Thayer School of Engineering

Dartmouth College

Hanover, New Hampshire 03755

Title of Paper: "An Algorithm for Resource Allocation Requiring Low Overhead Communication"

Submitted to: IEEE Transactions on Computers

Author Information:

Ann A. Marks
127 Cummings Hall
Dartmouth College
Hanover, New Hampshire 03755
(603) 646-2393

Paper Title: "An Algorithm for Resource Allocation Requiring Low Overhead Communication"

Submitted To: IEEE Transactions on Computers

Index Terms:

distributed computer systems, resource allocation, heuristic optimization technique, decentralized optimization technique

An Algorithm for Resource Allocation Requiring Low Overhead Communication

A heuristic algorithm for allocating resource units to sites in a distributed system is presented. Starting with a given allocation of sites, the algorithm performs a series of optimizations involving pairs of sites in an attempt to improve the worst pair-wise imbalance present in the system; termination occurs when no further improvement is possible. After outlining the general form of the algorithm, which effectively defines an entire family of algorithms, we present theoretical results that speak to the performance of the algorithm as measured in the number of optimizations that can be done, the amount of control communication required and the worst case inbalance of the resulting allocation. Subsequently, two particular algorithms in the family are given and the results of a simulation study of their performance is presented.

Index Terms - distributed computer systems, resource allocation, heuristic optimization technique, decentralized optimization technique

List of Figure Captions

Fig. 1. The resource allocation algorithm

Fig. 2. Distributed form of the resource allocation algorithm

Fig. 3. The mingreedy pair-wise optimization algorithm

Fig. 4. The maxgreedy pair-wise optimization algorithm

Fig. 5. Final maxΔ/mean for the centralized allocation

Fig. 6. Final maxΔ/mean for the semi-distrib. allocation

Fig. 7. Final maxΔ/mean for the fully distrib. allocation

Fig. 8. Comparison for final_maxΔ/mean for the centralized allocation

Fig. 9. Comparison for final_maxΔ/mean for the semi-distrib. allocation

Fig. 10. Comparison for final_maxΔ/mean for the fully distrib. allocation

Fig. 11. Comparison of number of opt's for exp.and uni. distrib's

Fig. 12. Comparison of the number of opt's for different initial allocations

Fig. 13. Comparison of the average number of opt's involving the same site

Fig. 14. Comparison of number of opt's for the centralized allocation

Fig. 15. Comparison of number of opt's for the semi-distrib. allocation

Fig. 16. Comparison of number of opt's for the fully distrib. allocation

Fig. 17. Comparison of ms/ss for the exp. and uni. distributions

Fig. 18. Comparison of ms/ss values for different initial allocations

Fig. 19. Comparison of ms/ss for the centralized allocation

Fig. 20. Comparison of ms/ss for the semi-distrib. allocation

Fig. 21. Comparison of ms/ss for the fully distrib. allocation

I. Introduction

Distributed systems inherently posess many advantages when contrasted with centralized systems. These include the potential to increase performance by use of parallelism, the ability to degrade gracefully and to be expanded incrementally, and the possibility of sharing expensive resources. Realizing these advantages requires answering a number of interesting questions. Among these is the question of assigning resources, such as databases and programs, to sites to achieve good performance. This question is important because it affects the amount of parallelism that can be achieved and determines how much communication must take place to access distributed data and to run distributed programs. In this paper, we present a family of algorithms for allocating resources to sites. These algorithms are novel in that they can be implemented with relatively little overhead communication for algorithm control. Thus, we say that these algorithms are decentral.

Because system control information in a distributed system is spread over all the sites in the system, exchange of information between different sites is necessary to perform nearly all control functions and to provide distributed services to users. Typically, the control information passed between the pieces of a distributed operating system describes the state of some object in the system. As such, this communication

constitutes an overhead which must be controlled if good performance is to be realized. There are two reasons why the amount of this communication must be limited. First, communication takes time, which can lower response time and throughput; also, it uses communication bandwidth, often a scarce resource in a distributed system, that might otherwise be devoted to user communication.

A number of different studies have examined different versions of the resource allocation problem. They are differentiated by the factors taken into consideration, the kind of resources being allocated and the nature of the algorithm used to assign resources to sites. Two major kinds of resource types that have been studied: information, of which databases are a prime example, and programs. Different issues are important for each. For example, database allocation[3, 5, 7, 11, 13, 14] must take into account the large expense incurred in processing update transactions. Here cost is incurred because, in general, some form of Byzantine agreement must be used to ensure the consistency of all copies of the same datum within the system. When allocating processing tasks to processors[1, 2, 4, 6, 12, 13, 15-19], there are two prime costs that have to be considered: the execution cost of the tasks, and the inter-process communication costs. Also, there are a number of other issues that can be included: constraints on memory space, deadlines, the cost of algorithm failure and associated restart. Finally, the policies employed in the underlying computer network may also be considered, e.g. the effect of dynamic

routing in a datagram-based network.

Previous algorithms for allocation resources in distributed systems fall into three major categories. First are the graph theoretic[19] which have been used to optimize the assignment of communicating tasks to processors. The system of processes is represented by a graph whose nodes represent either a process or a processor. Edges between process nodes are weighted with the communication cost incurred between the two associated processes. Edges between a process node and a processor node reflect execution costs. The optimal assignment corresponds to the minimum n-way cutset of this graph, where n is the number of processors. Extensions of this basic approach allow allocation to be determined based on critical load factors[17], to incorporate memory space limitations[15], and to account for the cost of relocating processes on a dynamic basis[2,18].

Programming methodology, e.g. linear and integer, has been used to find assignments of resources to sites[3, 5, 11-13, 16]. Typically, an objective function is to be minimized subject to constraints on other costs and resource availability. For example, the objective function might measure the cost of process execution plus inter-process communication costs. The constraints might capture limitations of available memory at the different sites. Solutions can be found employing standard approaches depending on the form of the cost function and constraints.

The third class of algorithms used for resource allocation are

heuristic in nature[1, 6, 7, 9, 14]. These algorithms are employed in two cases. First, they can be used when acceptable rather than optimal allocations are desired. Second, sometimes there is no feasible algorithm for computing optimal assignments of resources to sites because the optimization problem to be solved is NP-Complete. In these cases, there is no recourse but to find approximate solutions. Typically, heuristics are greedy and of an iterative form. Usually, one resource unit is allocated per iteration of the algorithm in a way to minimize or maximize some cost function.

Three other works are noteworthy. First, probabalitic models[4] have also been used to model the computational activity taking place in a distributed system; an iterative search algorithm is used to find allocations. The second work[9] uses exactly the same model as the graph theoretic one presented in[19]; however, the solution technique is parametric. Finally, the scheduling policies used by distributed servers changes the balance of load in a distributed system; the effects of different policies are presented in [20].

The family of algorithms presented in this paper is heuristic because the resource allocation problem, as reflected in its performance metric, is NP-Complete. The general form of the algorithm is greedy, and hence, iterative. All members of the family use the same rule to control iteration. Different algorithms within the family use different rules to control what allocations are made in each iteration but subject only to one

constraint. The two particular algorithms within this family which are presented in later sections of the paper, employ greedy methods to make allocations.

In contrast to previously reported algorithms, the family of algorithms studied in this paper are decentral. This means that the algorithm executes concurrently at all sites in the system and that each site runs with its own local state information plus a small amount of state information from the other sites. The techniques outlined above all require complete information on the entire system. To run them requires that all state information be sent to a single site which runs the allocation algorithm and then transmits the new assignment to all other sites. This is potentially expensive and inherently unreliable. Although such a technique can be used successfully to design a system or on a static basis, if used dynamically or when the size of the over-all system state is very large, unacceptable delay and communication may be incurred. However, optimal assignments can be found if the resource allocation problem can be solved for optimal assignments. Our algorithm family, because it is decentral and because the underlying problem is NP-Complete, sacrifices optimality of the resulting assignment for algorithm overhead.

The rest of this paper is organized as follows. In section II, we present the notation and terminology that will be used throughout the rest of the paper and state the resource allocation problem that we solve. The

general form of the algorithm is given in Section III along with theoretical results that speak to the performance of the algorithm family. In Section IV, two particular algorithms within the family are introduced. A simulation study of these two algorithms and the results of this study are given in Section V. Finally, in Section VI we summarize the results.

II. Notation and Problem Statement

In this section, we present our notation and then formally state the resource allocation problem studied in this paper.

Let:

S be the total number of sites in the system,

U be the total number of resource units in the entire system,

i, j, k and l are indices of sites,

$\mathcal{U}$ be the set of all resource units in the system,

u be some resource unit in $\mathcal{U}$,

$\mathcal{A}_i$ be the set of resource units assigned to site i

$= \{ u \mid u \text{ assigned to site } i \}$.

An assignment of units to sites is a partition of $\mathcal{U}$, i.e. $\bigwedge_{i \neq j} (\mathcal{A}_i) = \emptyset$ and $\bigcup_{i=1}^{S} \mathcal{A}_i = \mathcal{U}$

Let:

s_u be the size of the resource unit u,

S_i be the total resource unit size of the units assigned to site i

$$= \sum_{u \in \mathcal{A}_i} s_u$$

$\Delta_{i,j}$ be the difference in site size between sites i and j

$= | S_i - S_j |$, and let

$\max\Delta = \max_{1 \le i,j \le S \text{ and } i \ne j}\{\Delta_{i,j}\}$.

We now formally state the resource allocation problem.

Problem Statement : Given U, M and a set of resource units, $\mathfrak{U}$, with associated sizes, find the assignment, given by $A_1, \ldots, A_S$, that minimizes the maxΔ.

The metric used to measure the balance of an allocation is the maxΔ, which points out the worst imbalance in the system between any pair of sites. As such, it is a very conservative measure of the unbalance present in an allocation.

This problem is NP-Complete. It can be cast as a decision problem:

Given U, M and a set of resource units, $\mathfrak{U}$, with associated sizes and some $\partial > 0$, find the assignment, $A_1, \ldots, A_S$ such that max$\Delta \le \partial$.

For two sites and $\partial = 0$ (i.e. exact balance) this is the 2-partition problem[8].

The NP-Completeness of the problem implies that there is no known algorithm for finding assignments that runs in less than deterministic exponential time. In effect, the combinatorial structure of the problem is so great that the only way to find the optimal assignment is to enumerate and calculate the maxΔ of all the possible S^U assignments and then to pick the one with minimum maxΔ. For this reason, any realistic algorithm will have to be heuristic in nature finding acceptable assignments that may not necessarily be optimal.

III. Algorithm

In this section, we present the general form of the resource allocation algorithm and then establish some theoretical results on its performance. The theoretical results address several performance metrics: running time as measured in terms of numbers of iterations, the balance of the final allocation and how much control communication is needed.

A. Algorithm

The general form of the algorithm is: Start with whatever allocation happens to exist when the algorithm begins running. Then attempt to improve the balance of the allocation by iteratively selecting pairs of sites, in greedy fashion, to optimize. Having selected a pair of sites, optimization of this pair involves moving resource units from one site to other until the two sites meet the balance property presented below. Iteration terminates when all pairs of sites fulfill this balance property. Note that because the algorithm begins with whatever assignment it finds, the resulting allocation depends on the initial assignment. This implies that the final allocation may not be optimal.

We need a way to decide when to stop moving resource units from one site to the other. Also, two other rules are needed: one to identify the pair of sites to optimize next, and one to determine when the system is

balanced and, hence, when iteration should terminate. The definition of pair-wise site balance serves as the rule for both.

Definition: Let i and j be two sites with site sizes S_i and S_j. If $S_i > S_j$ then site i is said to be the big site and site j is said to be the small site. □

Definition: Pair-Wise Balance. Let i and j be two sites with site sizes S_i and S_j and let site i be the big site, i.e. $S_i > S_j$. Sites i and j are balanced whenever there is no resource unit u assigned to site i with size s_u such that:

$$S_i - s_u > S_j + s_u. \square$$

This definition says that moving any resource unit from the big site, i, to the small site, j, reverses the roles of the big and small sites, i.e. the big site, i, becomes the small site, and the small site, j, becomes the big site. The next definition specifies when two sites are not in balance.

Definition: Pair-Wise Unbalance. Let i and j be two sites with site sizes S_i and Sj and let site i be the big site. Let u be some resource unit assigned to site i and let s_u be its size. Sites i and j are said to be unbalanced whenever:

$$S_i - s_u > S_j + s_u. \square$$

The definition of pair-wise unbalance says that there is a resource unit on the big site which can be moved to the small site and leave the big

site the big site. When two sites are unbalanced, their $\Delta_{i,j}$ can be lowered by moving one or more resource units from the big site to the small site. Note that the definition of balance is is very slack. In the algorithm, balance is used to terminate the movement of resource units from the big site to the small site in a pair-wise optimization. Thus, the site, which is initially the big site, can not become the small site even if doing so would result in a lower $\Delta_{i,j}$. The reason for using such a slack criterion for balance is that it results in less total unit size being moved, which constitutes an over-head cost associated with running the algorithm. However, the definition of balance could be recast along the lines indicated above and the theoretical results given below could still be derived with the appropriate changes. The algorithm is given in Figure 1.

<u>while(</u> there is any pair of unbalanced sites)

let i and j be a pair of unbalanced sites with site sizes S_i and S_j such that $\Delta_{i,j} = \max\Delta$.

<u>while(</u> site i and j are unbalanced)

move a resource unit u, currently assigned to site i and which has size s_u, to site j provided:

$S_i - s_u > S_j + s_u$, i.e. reallocation leave sites i the big site.

Update $S_i = S_i - s_u$, $S_j = S_j + s_u$, $\mathcal{A}_i = \mathcal{A}_i - \{u\}$, and $\mathcal{A}_j = \mathcal{A}_j \cup \{u\}$.

<u>end</u>

<u>end</u>

Fig. 1. The resource allocation algorithm

The algorithm consists of two nested loops. The outer loop controls iteration, which continues as long as there is a pair of unbalanced sites. In each iteration, a pair of sites is selected, and this pair must have a Δ which equals the current maxΔ (in case of a tie on the maxΔ an agreed upon arbitration rule is used to pick the pair to optimize). Once inside the outer loop, there must be a pair of sites that is unbalanced. Hence, there is at least one unit on the big site which can be moved. The inner loop, which performs the pair-wise optimization, moves units from the big site to the small site until the two sites become balanced. Note that there may be more than one resource unit on the big site which can be moved at any point during the pair-wise optimization; for now, no rule is specified for how to pick between such units.

We present the following lemma that addresses the relationship

between the site sizes of the pair of sites involved in a pair-wise optimization both before and after the optimization.

Site Size Lemma: Let i and j be two sites and let site i be the big site. Furthermore, assume that sites i and j are unbalanced and are the ones selected for a pair-wise optimization. Let ms be the total resource unit size moved from site i to site j by the pair-wise optimization, i.e

$ms = \Sigma_{\{u \mid \text{moved from site } i \text{ to } j\}} s_u$

If S_i and S_j are the sizes of site i and j before the pair-wise optimization and S°_i and S°_j are the sizes after the pair-wise optimization then $S_i > S^{\circ}_i = S_i - ms > S^{\circ}_j = S_j + ms > S_j$.

Proof: Because site i is the big site, initially $S_i > S_j$. If only one resource unit u, with size s_u, is moved from site i to site j, then $S_i > S^{\circ}_i = S_i - s_u$ and $S^{\circ}_j = S_j + s_u > S_j$, i.e. the new site size of site i is smaller than the old and the new site size of j is larger than the old. It follows from the definition of imbalance and the way imbalance is used by the algorithm that the big site can not become the small site, i.e. $S^{\circ}_i > S^{\circ}_j$. By induction on the number of units moved, the result follows directly. □

B. Theoretical Results

We now present results that establish how well the algorithm performs. Specifically, we establish a bound on the number of optimizations that can be done by the algorithm. From this result, we can easily bound total resource unit size moved, and bound the amount of control communication. A weak bound on the final maxΔ is also given.

To show how many times the algorithm can iterate, we examine the sequence of maxΔ's generated as the algortihm runs. First we establish that this sequence is non-increasing. Then we bound how long the maxΔ can remain the same. With a bound on the number of times the maxΔ can decrease, the number of iterations made by the algorithm can be bounded.

maxΔ site size lemma: Let sites i and j be the ones selected for optimization, i.e. $\Delta_{i,j}$ = maxΔ. There can be no pair of sites, k and l, with $S_k > S_i$ and $S_l < S_j$.

Proof: The proof is by contradiction on the possible relative values for S_k and S_l.

First, if $S_k > S_i$ then $\Delta_{k,j} > \Delta_{i,j}$ which contradicts the assumption that $\Delta_{i,j}$ = maxΔ.

Second, if $S_l < S_j$ then $\Delta_{i,l} > \Delta_{i,j}$ again contradicting the assumption that $\Delta_{i,j}$ = maxΔ.

Third, if $S_k > S_i$ and $S_l < S_j$ then $\Delta_{k,l} > \Delta_{i,j}$ contradicting the

assumption that $\Delta_{i,j} = \max\Delta$. □

To establish that the maxΔ sequence is non-decreasing, we introduce the following notation.

Notation: Let $\max\Delta_n$ be the n-th maxΔ which occurs as the algorithm runs, and let N be the total number of optimizations done. If $\max\Delta_0$ is the initial maxΔ then the sequence of maxΔ's which occurs as the algiorithm runs is $\max\Delta_0, \max\Delta_1, \max\Delta_2, \ldots, \max\Delta_N$. □

Non-Increasing Lemma: If $\max\Delta_0, \max\Delta_1, \ldots, \max\Delta_N$ is the sequence of maxΔ's generated by the algorithm then $\max\Delta_n \geq \max\Delta_{n+1}$, $0 \leq n \leq N-1$.

Proof: Throughout this proof, we use primed symbols to indicate values after the optimization and unprimed symbols to indicate values before. Let us assume that the sites involved in the optimization are i and j and let site i be the big site.

First consider two sites, k and l other than i and j. From the maxΔ site size lemma, we know that $\Delta_{k,l} \leq \Delta_{i,j}$. Because sites k and l are not involved in the optimization, $\Delta'_{k,l} = \Delta_{k,l}$ and, thus, $\Delta'_{k,l} \leq \Delta_{i,j}$. Hence, if sites k and l were the next pair chosen for optimization, then the next maxΔ could not be larger than the previous one.

Now consider the sites i and j. From the site size lemma, we know that $\Delta'_{i,j} < \Delta_{i,j}$. We need only consider the $\Delta_{i,k}$'s $(k \neq j)$ and the $\Delta_{k,j}$'s $(k \neq i)$ and show that $\Delta'_{i,k} < \Delta_{i,j}$ and that $\Delta'_{k,j} < \Delta_{i,j}$. It suffices to argue for site

i since the argument for site j is similar.

There are two cases of interest, namely those sites k with $S_k > S'_i$ and those sites k with $S_k < S'_i$ (Note that, since site k was not involved in the optimization, $S'_k = S_k$.) First, if $S'_i > S_k$ then $\Delta_{i,k} = S_i - S_k > S'_i - S_k = \Delta'_{i,k}$ because, by the site size lemma $S_i > S'_i$.

The second case considers $S'_i < S_k$. If ms is the total resource unit size moved from i to j then $\Delta'_{i,j} = S'_i - S'_j = (S_i - ms) - (S_j + ms) = S_i - S_j - 2ms = \Delta_{i,j} - 2ms$ or that $\Delta'_{i,j} + 2ms = \Delta_{i,j}$. But this implies that $\Delta_{i,j} > 2ms > ms$. We can show that $ms = \Delta_{i,k} + \Delta'_{i,k} = (S_i - S_k) + (S_k - S'_i) = S_i - S'_i = S_i - (S_i - ms) = ms$ where we note that $S_i \geq S_k$ by the maxΔ site size lemma. Hence the result that $\Delta_{i,j} > \Delta'_{i,k}$. □

We now know that the maxΔ sequence is non-increasing. This means that the values in this sequence can either remain the same or decrease. It is possible for the maxΔ sequence to include a series of values that are the same, i.e. the sequence can contain flat spots. Next we show that such a flat spot can be at most floor(S/2) long.

S/2 lemma: Let v be some value that occurs in the maxΔ sequence. There can be no more than floor(S/2) sequential repetitions of v in the maxΔ sequence.

Proof: This follows from the non-increasing lemma and the fact that there can be at most floor(S/2) pairs of sites that will give the same Δ

value v. To see this realize that at some point in time, n, there may be up to floor(S/2) pairs of sites whose $\Delta = \max\Delta_n$. Once such a pair of sites, say i and j, is involved in an optimization, the max site size lemma establishes that $\Delta'_{i,j} < \Delta_{i,j} = \max\Delta_n$. Also, when considered against any other site k, the max site size lemma says that $\Delta'_{i,k}, \Delta'_{j,k} < \Delta_{i,j} = \max\Delta_n$. Therefore sites i and j are not candidates for optimization as long as there are sites k and l with $\Delta_{k,l} = \max\Delta_n$. Therefore, after floor(S/2) optimizations the population of sites that could give a $\Delta = \max\Delta_n$ must have gone to zero. □

The lemma just proven says that the maxΔ sequence can be flat for only floor(S/2) optimizations. Flat spots in the maxΔ sequence must be separated by at least one optimization in which the maxΔ decreases. The next lemma establishes a bound on the number of such decreases.

Number of Decreases Lemma: The number of decreases occuring in the maxΔ sequence is O(U).

Proof: Between flat spots in the maxΔ sequence there must be at least one optimization in which the maxΔ decreases. At worst the maxΔ value can decrease by one. The number of decreases can easily be bounded. If TOTAL_SIZE = $\Sigma_{u \in \mathcal{U}} s_u$ is the total size of all resource units in the system then there can be at most TOTAL_SIZE such decreases. This corresponds to $\max\Delta_0$ = TOTAL_SIZE, i.e. all units are initially assigned to

the same site, and a $max\Delta_N = 0$, i.e. the final allocation is exactly balanced. If s_u is the largest unit size over all units then TOTAL_SIZE $\leq Us_u$. Therefore, the number of decreases is O(U).

Finally, by combining the last two lemmas we can bound the number of optimizations done by the algorithm.

<u>Theorem</u>: The number of optimizations done by the algorithm is O(SU).

<u>Proof</u>: At worst, the maxΔ sequence can be flat for floor(S/2) optimizations followed by a decrease of one. This sequence of repeats followed by a decrease, which has length floor(S/2) + 1, can be repeated only O(U) times. □

Since the resource unit size moved per optimization can easily be bounded, we can establish the following corrollary.

<u>Moved Size Corrollary</u>: The total resource unit size moved as the algorithm runs is $O(SU^2)$.

<u>Proof</u>: In each pair-wise optimization, at most TOTAL_SIZE units of resource can be moved and TOTAL_SIZE is O(U). Since the number of optimization is O(SU) the cumulative resource unit size moved as the algorithm runs is $O(SU^2)$. □

The final maxΔ can be very weakly bounded:

<u>Final maxΔ Theorem</u>: $max\Delta_N \leq 2maxs_u$ where $maxs_u$ is the size of the largest resource unit in the system.

<u>Proof</u>: This follows immediately from the fact that if the maxΔ >

$2maxs_u$ then there must be a pair of sites, i and j, which are not balanced. If such a pair existed then even the largest resource unit could be moved from the big site to the small site and leave the big site big. □

We now present a distributed form of the algorithm so that we can determine how much communication is needed for control purposes. Figure 2 shows the algorithm in a form suitable for distributed execution as it would be run at each site in the system.

```
/* Each site runs the following algorithm in parallel. */
/* initialization */
send my S_i and mins_u to all other sites;
/* mins_u = min {s_u} */
/*          u∈A_i     */
receive S_j's and mins_u's from all other sites;
do( when time to reallocate ) /* algorithm runs periodically */
        while( system not balanced )
        /* Using the mins_u's every site test for balance */
                determine the pair of sites to participate in pair-wise
                optimization
```

<u>if</u> I am the big site in the optimization with index i <u>then</u>

<u>until(</u> balance achieved between me and the small site, j)

send some resource unit, u, to the small site, j, such that $S_i - s_u > S_j + s_u$,

Set $S_i = S_i - s_u$, $S_j = S_j + s_u$,

$mins_{uj} = min(mins_{uj}, s_u)$ and $\mathcal{A}_i = \mathcal{A}_i - \{u\}$

<u>end</u>

send my final S_i and $mins_u$ to all other sites

<u>elseif</u> I am the small site, with index j <u>then</u>

<u>until(</u> balance achieved between me and the big site)

receive some resource unit, u with size s_u, from the big site,

Set $S_j = S_j + s_u$, $S_i = S_i - s_u$,

$mins_{ui} = min(mins_{ui}, s_u)$ and $\mathcal{A}_j = \mathcal{A}_j \cup \{u\}$

<u>end</u>

send my final S_j and $mins_u$ to all other sites

<u>else</u>

/* i am not involved in the optimization */

receive S's and $mins_u$'s from both sites

participating in the optimization

fi

end /* of while system not balanced */

end /* of when time to reallocate */

Fig. 2. Distributed form of the resource allocation algorithm

---

Theorem: The amount of control communication, i.e. that required to transmit the S_i's and $mins_u$'s, is $O(S^2U)$ when the underlying network does not support broadcasts or $O(SU)$ when the network does implement broadcasts.

Proof: The amount of control communication to send the S_i's and $mins_u$'s depends on whether the underlying net supports broadcasts. Thus, the analysis examines both cases. First, the initialization requires that each site send its current S_i and $mins_u$ to all other sites. This takes $O(S^2)$ without broadcasts or $O(S)$ with. Once the algorithm is running, without broadcasts, each optimization results in $O(S)$ messages being sent to transmit the new S_i's and $mins_u$'s at the end of every optimization. Alternatively, only two messages need be sent if the networks supports broadcasts. Thus, given that $O(SU)$ optimizations are needed to balance the

entire system, there will be $O(S^2U)$ control messages sent without broadcasts or $O(SU)$ sent with broadcasts. □

We now have established that the number of iterations done by the algorithm is $O(SU)$, that the total resource unit size moved is $O(S^2U)$, that the worst imbalance that can exist when the algorithm is done balancing the system is at most $2*maxs_u$ and that either $O(SU)$ or $O(S^2U)$ control messages are sent as the algorithm runs.

VI. Two Site Optimization Algorithms

In this section, we present two algorithms for selecting which resource unit to move in a pair-wise optimization. Both are greedy. In the next section, we outline a simulation study conducted using these two algorithms and present results of this study.

We have studied two greedy algorithms for performing a pair-wise optimization. One is greedy in a minimum sense; the other is greedy in a maximum way. The minimum greedy algorithm, which we will refer to as mingreedy, is given in Figure 3. The maximum greedy algorithm, maxgreedy, is shown in Figure 4. In both cases, we show only the loop needed to perform a pair-wise optimization. This loop would be embedded in the control loop shown in the previous section.

/* Let the sites involved in the optimization be i and j and let site i be the big site. Note that on the big site, i refers to this site and j to the small site. On the small site, j refers to this site and i to the big site. */

<u>if</u> This site is involved in the optimization <u>then</u>

    <u>while</u>($S_i - mins_{ui} > S_j + mins_{ui}$)

        /* while the two sites are not balanced */

        <u>if</u> This is the big site <u>then</u>

move my smallest resource unit, u with size $mins_u$

to the small site

update $S_i = S_i - mins_u$, $\mathcal{A}_i = \mathcal{A}_i - \{u\}$ and $S_j = S_j + mins_u$.

elseif This is the small site then

receive a resource unit, u with size s_u, from the big site

update $S_j = S_j + s_u$, $\mathcal{A}_j = \mathcal{A}_j \cup \{u\}$, and $S_i = S_i - mins_u$.

fi

end /* of while not balanced */

send my new S_i (or S_j) and $mins_u$ to all sites

else

/* This site was not involved in the optimization */

receive the new S's and $mins_u$'s from both the big and small sites

fi

Fig. 3. The mingreedy pair-wise optimization algorithm

Note that, in the mingreedy algorithm, the resource unit moved from the big site to the small site is always the one with smallest size among those currently allocated to the big site. In contrast, the max greedy algorithm attempts at each iteration to come as close to filling the Δ between the big and small sites. This it accomplishes by chosing to move the unit, currently allocated to the big site, whose size most nearly fills the $\Delta_{i,j}$ without letting the big site become the small site.

/* Let the sites involved in the optimization be i and j and let site i be the big site. Note that on the big site, i refers to this site and j to the small site. On the small site, j refers to this site and i to the big site. */

<u>if</u> This site was involved in the optimization <u>then</u>

<u>while</u>(S_i - $mins_{ui} > S_j + mins_{ui}$)

/* While the two sites are not balanced */

<u>if</u> This is the big site <u>then</u>

move the largest resource unit, u with size s_u, to the small site but preserve $S_i - s_u > S_j + s_u$

update $S_i = S_i - s_u$, $\mathcal{A}_i = \mathcal{A}_i - \{u\}$ and $S_j = S_j + s_u$.

<u>elseif</u> This is the small site <u>then</u>

receive a resource unit, u with size s_u, from the

big site

update $S_j = S_j + s_u$, $\mathcal{A}_j = \mathcal{A}_j \cup \{u\}$ and $S_i = S_i - s_u$.

fi

end /* of while not balanced */

send my new S_i (or S_j) and $mins_u$ to all sites

else

/* This site was not involved in the optimization */

receive the new S's and $mins_u$'s from the big and small sites

fi

Fig. 4. The maxgreedy pair-wise optimization algorithm

Note that the overall control structure for both algorithms is identical. Both have a while loop that controls iteration. After the loop terminates, the big and small sites send their new site size and $mins_u$ to all other sites. The sites not involved in the optimization only wait to receive the new site sizes and $mins_u$'s. The only difference in these two algorithms is how the unit to move from the big site to the small is selected.

V. Simulation Results

In this section, we investigate the performance of both the max and mingreedy algorithms given in the previous section by way of a simulation study. First, we outline the study by presenting the parameters of the problem and the metrics used to measure and evaluate performance of the algorithms. Subsequently, the results of the study are presented.

A. Simulation Study

Several different parameters can potentially affect the performance of the algorithms. These include:

the statistical distribution of resource unit sizes,

the initial allocation of resource units to sites,

the number of sites and

the number of resource units.

The metrics used to measure algorithm performance are:

the final maxΔ,

the total number of optimizations performed by the algorithm and

the total resource unit size moved to balance the system.

To compare the performance of the two algorithms relative to one another, we will use delta-percents:

Δ% = (losing_algorithm's_value -

winning_algorithm's_value)/losing_algorithm's_value. □

This formulation is reasonable because, for the performance metrics

given above, small values always indicate better performance.

As we will see in examing the results, sometimes one algorithm will do better than the other on some but not all metrics. In these cases, it is important to understand on what parameter values and for what performance metrics an algorithm does better or worse. Other questions address how the performance metrics vary as the parameters are changed.

We have conducted the simulation study using two different distributions of resource unit sizes: exponential and uniform. The exponential is interesting because it is an accurate model of resource size for some kinds of resources, e.g. file size distribution. The uniform serves as a basis for comparison so that we may gain some insight into how important the resource size distribution is in determining performance.

Three different initial allocations of resource units to sites were studied. In the first allocation, which we will refer to as centralized, all resource units were initially assigned to the same site. We have chosen to simulate for this allocation because it should force the algorithm to perform the greatest number of optimizations and to move the most cumulative resource unit size. Of course, if the initial distribution is centralized, a centralized algorithm can easily be run at much less over-head. The second allocation studied is semi-distributed. Here equal numbers of resource units were placed on half the sites (when the number of sites did not evenly divide the number of units, one site had remainder(U/S) units allocated to it). Finally, a fully distributed initial allocation was also studied in which equal numbers of units were

allocated to all sites (one site might be allocated remainder(U/S) units).

To study the effect of the number of sites, we simulated using three different numbers of sites: 6, 12 and 20. The effects of varying the number of units were studied by simulating with five different numbers of units: 30, 60, 120, 180 and 240.

Finally, in conducting these simulations, we chose a maximum unit size of 30 for the uniform resource unit distribution. For the exponential distribution, the mean was set to be 50 and the maximum allowable unit size was 5000. Studies with both the uniform and exponential distribution showed that the choice of distribution parameters did not affect the nature of the simulation results. Specifically, the number of optimizations stayed constant regardless of distribution parameters. Final maxΔ and total moved size varied linearly in the mean of the distribution. Lastly, at all data points in the study, a total of 3000 simulations were performed in order to accurately measure the mean value of the metrics.Throughout the following sections, all results and numbers given are based on the mean value of the performance metrics.

B. Simulation Results

We now look in turn at each performance metric and answer how unit size statistics, initial distribution, number of units, and number of sites affected algorithm performance. Also, we present the comparison between

the min and maxgreedy algorithms.

B.1 Final MaxΔ

To evaluate the performance of the algorithms based on final maxΔ, we have chosen to normalize the actual final maxΔ's by dividing by the mean of distribution. This allows us to compare the results for the uniform and exponential distributions in a meaningful way because such a measure normalizes the results. Examining the normalized values, we found that the uniform and exponential distributions had results with the same form. However, there was a difference in the actual values for the Final_MaxΔ/Mean. The exponential gave a range of 0.04 to 3.0, and the uniform distribution showed a range of 0.08 to 1.88. Typically, the values for the uniform distribution were about 1/3 that of the exponential. Hence, the statistical distribution of unit sizes had no effect on the form of the results, but did affect the particular values for the normalized final maxΔ. Therefore, it suffices to present and discuss the results for the exponential resource unit size distribution.

Figures 5 to 7 present the results for final maxΔ for the three different initial allocations. These plots show that the initial distribution, and the number of sites always affect final maxΔ performance. Specifically, the centralized allocation gives rise to very erratic behavior which depends on the number of sites: For 6 sites, the final maxΔ decreases as the number of units is increased. At 12 sites, the final maxΔ peaks at 120 units. For 20 sites, the final maxΔ goes through a minimum at 60 units. In the semi-centralized initial allocation, the final maxΔ is essentially flat for a given number of sites, and therefore

insensitive to number of units. For fully distributed initial allocation, the final maxΔ decreases sharply as number of units increases. Also, note that for a given algorithm, performance tends to worsen as the number of sites increases.

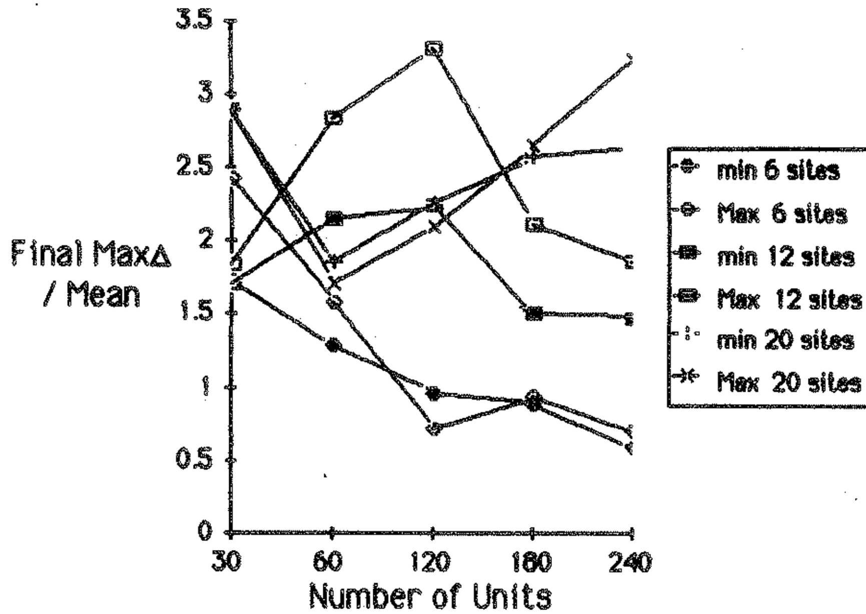

Fig. 5. Final maxΔ/mean for the centralized allocation

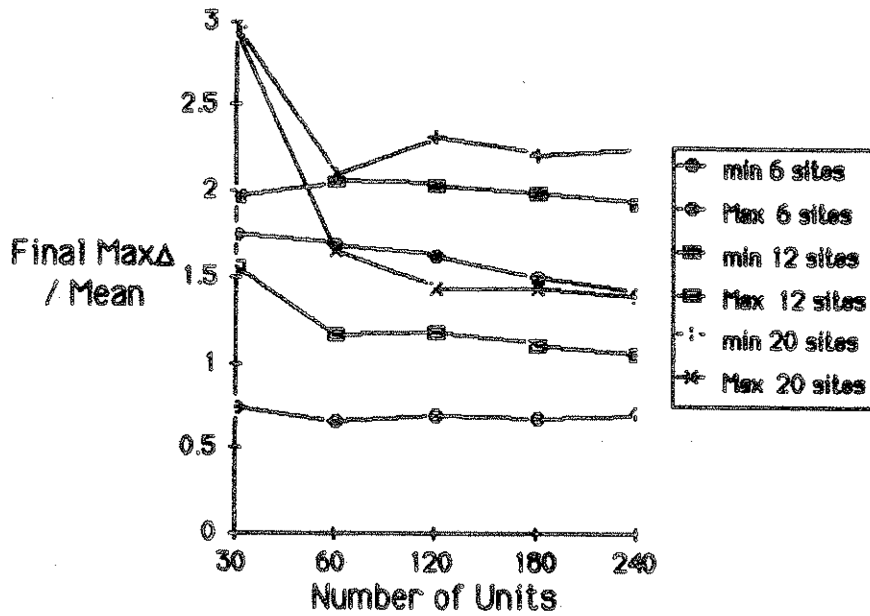

Fig. 6. Final maxΔ/mean for the semi-distrib. allocation

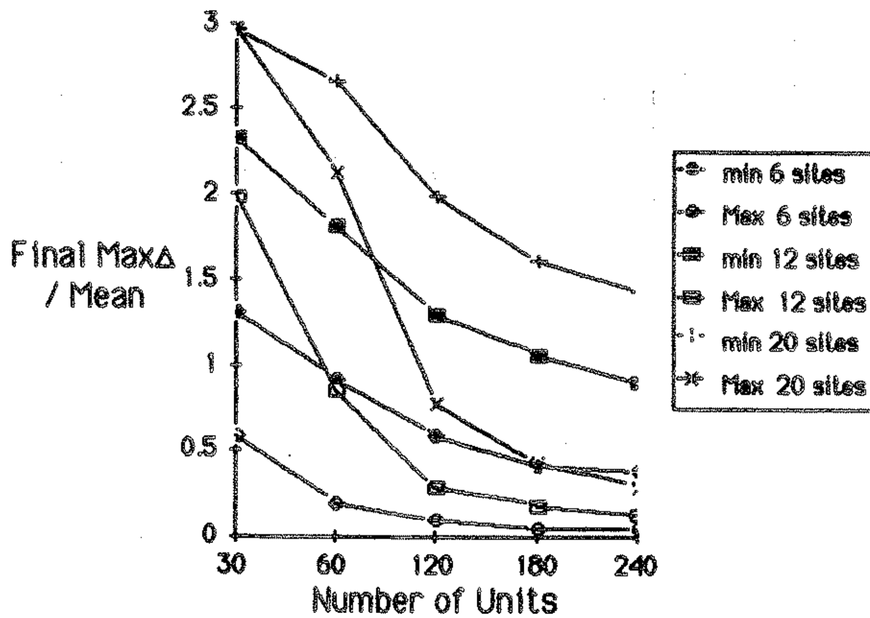

Fig. 7. Final maxΔ/mean for the fully distrib. allocation

Figures 8 to 10 show the delta-percents that compare the final maxΔ's of the min and maxgreedy algorithms. Only for centralized initial allocation did the mingreedy algorithm perform better than the maxgreedy but by relatively small amounts: 31.90% to -31.42% (the -31.42% was a one point anamoly). In contrast, for the semi- and fully distributed initial allocations, maxgreedy won handily particularly for larger numbers of units.

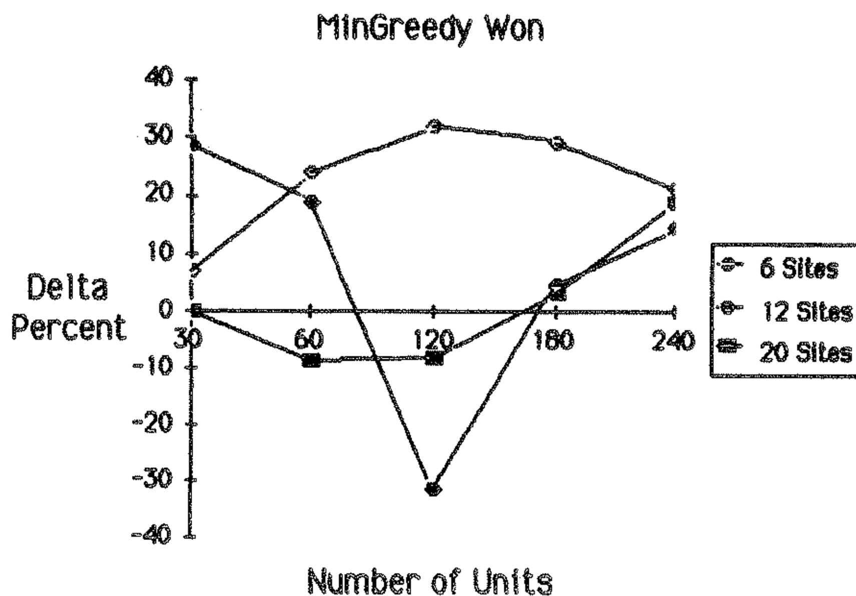

Fig. 8. Comparison for Final maxΔ/mean for the centralized allocation

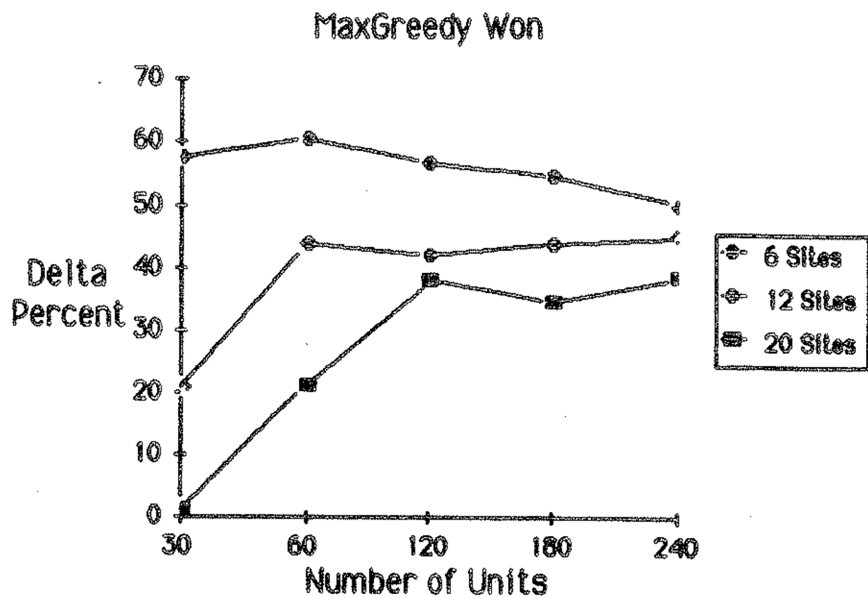

Fig. 9. Comparison for Final maxΔ/mean for the semi-distrib. allocation

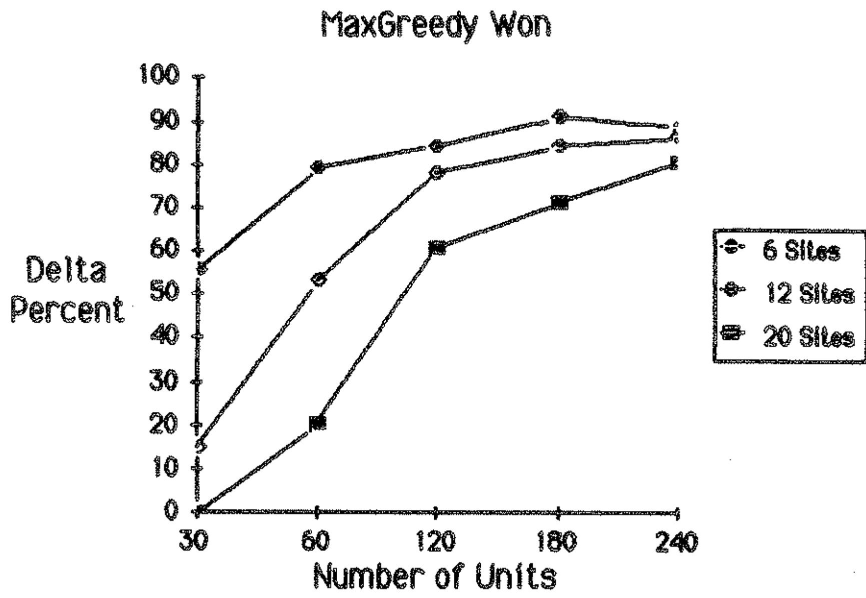

Fig. 10. Comparison for Final maxΔ/mean for the fully distrib. allocation

B.2 Number of Optimizations

We now compare the algorithms' performance on the number of optimizations. We present the actual number of optimizations done as the simulations ran. Figure 11, which gives typical data, shows that the statistical distribution of unit sizes does not appreciably affect the number of optimizations (Figure 11 data are for the centralized initial allocation and the maxgreedy algorithm.) Thus, we present and discuss the results for the exponential resource unit size distribution.

Number of Sites	Uniform Number of Units 30	240	Exponential Number of Units 30	240
6	9.34	15.73	9.20	15.71
12	13.95	26.18	14.18	26.04
20	21.96	32.38	22.02	33.31

Fig. 11. Comparison of number of opt's for exp.and uni. distrib's

For all the points in the parameter space, we found that the number of optimizations done was a nearly linear function of the number of units. Figure 12 displays the range of the number of optimization values for exponential distribution and the maxgreedy algorithm. At a fixed number of sites and units, as the initial allocation becomes more distributed, the

number of optimizations decreases. For a given initial distribution, as the number of sites is increased, the number of optimizations goes up. For a given number of sites and for a particular initial allocation, the number of optimizations increases as the number of units is increased. Looking at the changes in the number of optimizations for a given initial allocation, we note that the number of sites plays a larger role in determining the number of optimizations than does the number of units. This is illustrated in Figure 13 below, which shows the average number of times a site is involved in an optimization (= [2*Number of Optimizations]/Number of Sites, where the 2 reflects the fact that every optimization involves two sites). Figure 13 also shows that the algorithm becomes more efficient as the number of sites increases as evidenced by the relatively low average number of times a site is involved in an optimization (The data in Figure 13 are for the exponential resource unit size distribution and the maxgreedy algorithm.)

Number of Sites	Centralized Number of Units 30	Centralized Number of Units 240	Semi-Distributed Number of Units 30	Semi-Distributed Number of Units 240	Fully-Distributed Number of Units 30	Fully-Distributed Number of Units 240
6	9.20	15.71	7.13	10.33	4.78	9.54
12	14.18	26.04	9.79	19.57	7.54	15.41
20	22.02	33.31	14.43	29.69	8.62	21.88

Fig. 12. Comparison of the number of opt's for different initial allocations

Number of Sites	Centralized Number of Units 30	240	Semi-Distributed Number of Units 30	240	Fully-Distributed Number of Units 30	240
6	3.07	5.24	2.38	3.44	1.59	3.18
12	2.36	4.34	1.63	3.26	1.26	2.57
20	2.20	3.33	1.44	2.97	0.86	2.19

Fig. 13. Comparison of the average number of opt's involving the same site

Overall, the mingreedy algorithm did fewer optimizations than the maxgreedy as Figures 14 to 16 show. Note that as the initial allocation becomes more distributed, mingreedy's performance advantage becomes more significant. Also, this advantage usually increases as the number of sites is increased (except for the centralized initial distribution where advantage goes down as the number of units is increased). Finally, the number of sites seems to have differing effects. For the fully distributed initial allocation, 6 sites resulted in better performance than 12 sites which in turn showed better performance than 20 sites. In contrast, for the semi-distributed initial allocation, above 120 units there was very little difference in the effect of the number of sites; below 120 units, performance was always best at 6 sites. The centralized initial allocation

data only show that mingreedy tends to do better than maxgreedy but not dramatically. In fact, there are points at which maxgreedy does better but not significantly. For the most part, mingreedy performs better at low numbers of units; with higher of numbers of units, either maxgreedy does better by up to about 6% or its advantage is very small (2% at best).

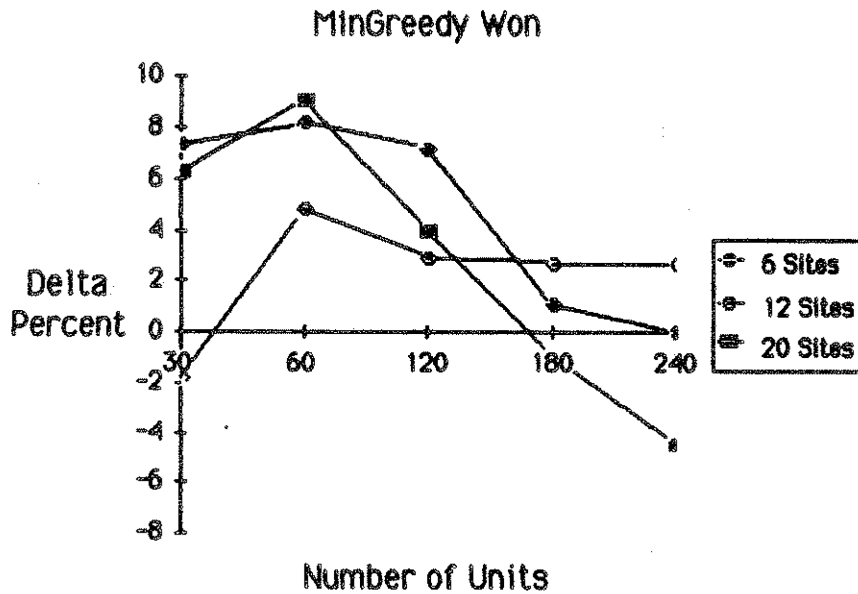

Fig. 14. Comparison of number of opt's for the centralized allocation

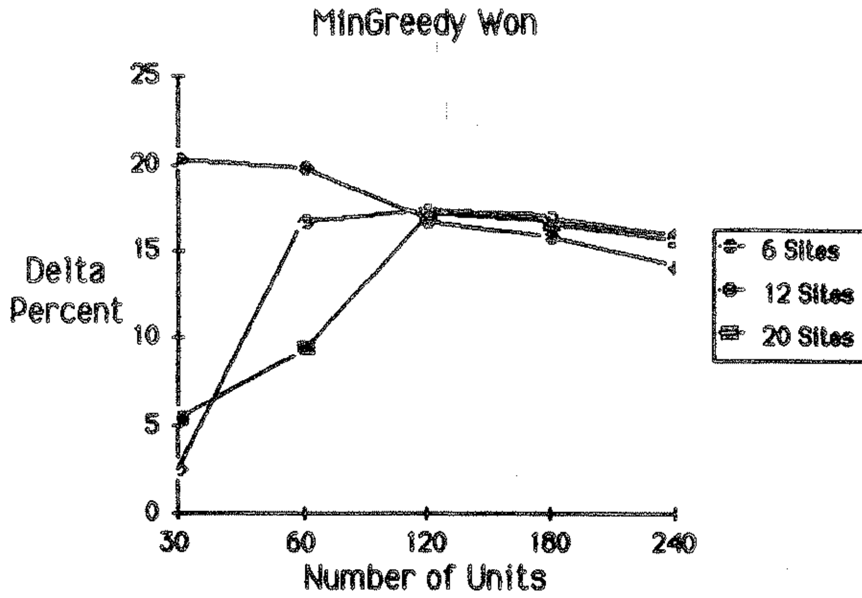

Fig. 15. Comparison of number of opt's for the semi-distrib. allocation

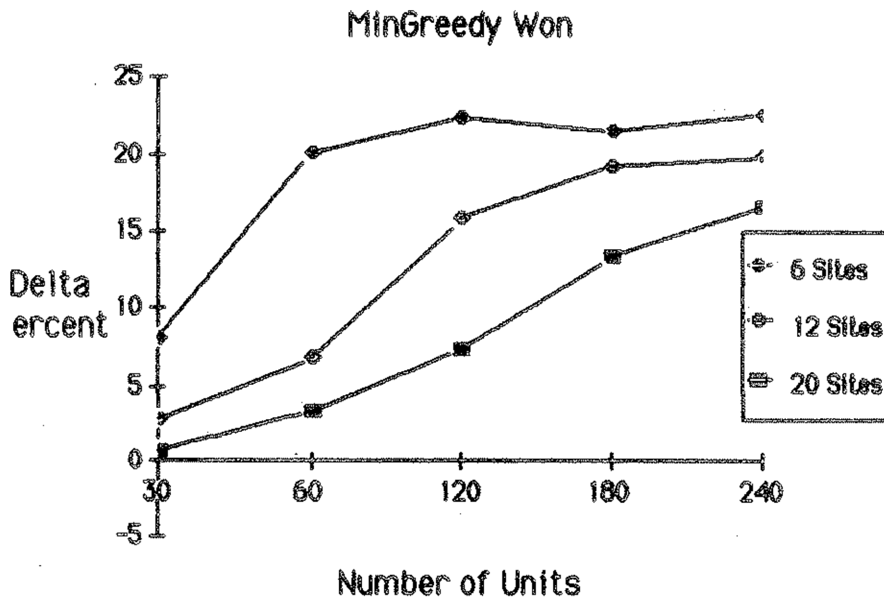

Fig. 16. Comparison of number of opt's for the fully distrib. allocation

B.3 Moved Size

As each simulation ran, we kept track of the total resource unit size moved. To be able to compare results from different runs, we have normalized the mean of the total resource unit size moved (ms) by dividing by the average total resource unit size in the system (ss), forming ms/ss (moved size/system size). Figure 17 shows that the statistical distribution of resource unit sizes has little effect on ms/ss (These data are for the centralized initial allocation and the maxgreedy algorithm.) Therefore, we present the results for the exponential distribution only.

	Uniform		Exponential	
Number	Number of Units		Number of Units	
of Sites	30	240	30	240
6	1.38	1.47	1.33	1.47
12	1.73	1.93	1.70	1.92
20	2.06	2.22	1.94	2.21

Fig. 17. Comparison of ms/ss for the exp. and uni. distributions

The data in Figure 18 show the variations induced by the initial distribution, number of sites and number of units for the maxgreedy algorithm (These data are for the exponential resource unit size distribution, and the maxgreedy algorithm.) As expected, the more

distributed the initial allocation, the less moved. Also, for the centralized and semi-distributed initial allocations, the ms/ss is essentially flat in the number of units; for the fully distributed case, the ms/ss decreases as the number of units is increased. Only for the centralized initial allocation did ms/ss increase as number of sites increases.

Number	Centralized Number of Units		Semi-Distrib Number of Units		Fully-Distrib Number of Units	
of Sites	30	240	30	240	30	240
6	1.33	1.40	.57	.54	.14	.06
12	1.70	1.79	.43	.56	.15	.15
20	1.94	2.21	.51	.57	.12	.12

Fig. 18. Comparison of ms/ss values for different initial allocations

Figures 19-21show that the mingreedy algorithm performed better. Mingreedy did better for the more distributed allocations where it did so by better than 10-15% consistently. At 6 sites, mingreedy's advantage always erodes as the number of units is increased. At 12 and 20 sites, no clear pattern emerges.

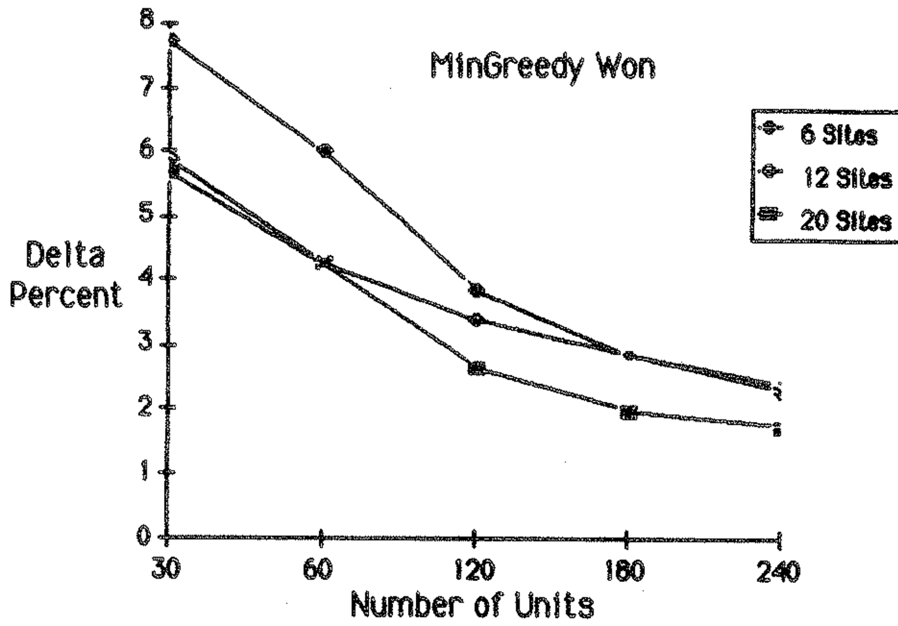

Fig. 19. Comparison of ms/ss for the centralized allocation

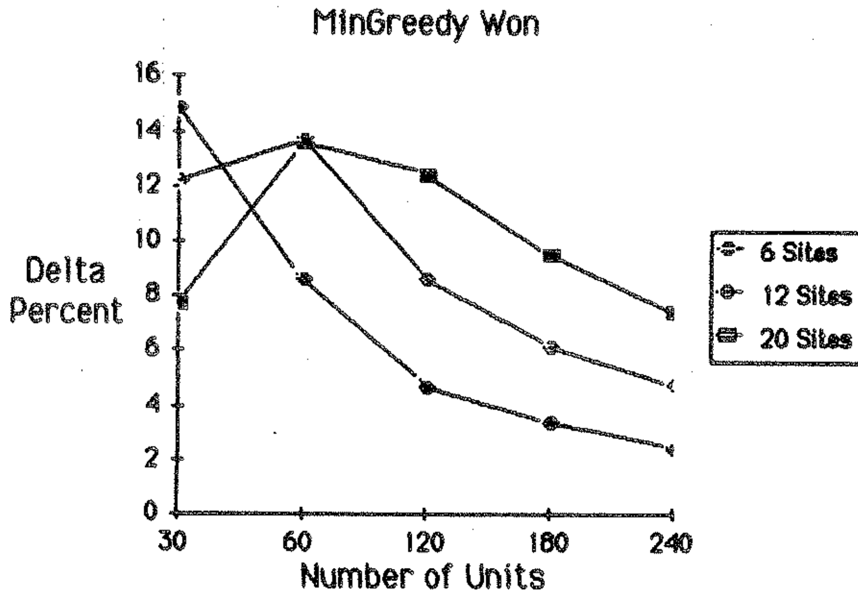

Fig. 20. Comparison of ms/ss for the semi-distrib. allocation

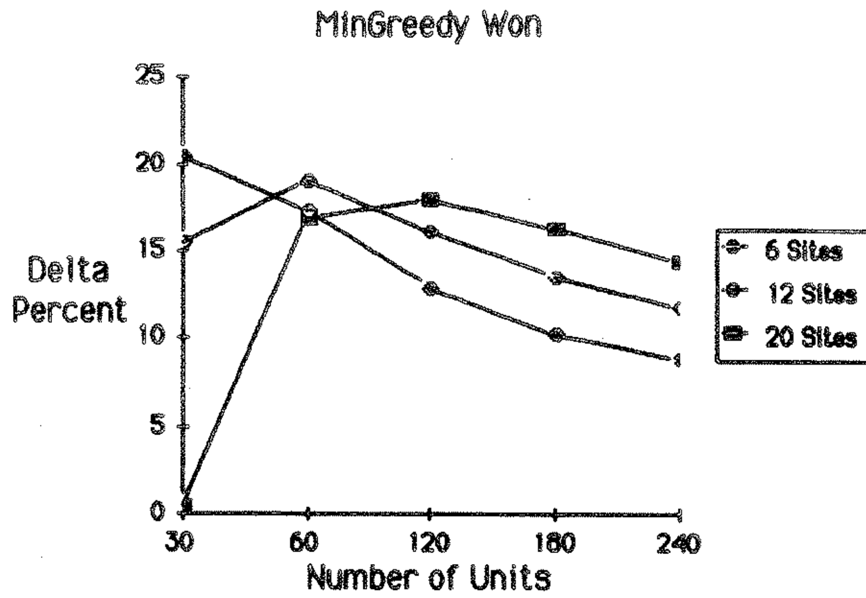

Fig. 21. Comparison of ms/ss for the fully distrib. allocation

VI. Conclusions

A family of algorithms for resource allocation in distributed systems has been presented in this paper. The underlying formulation of the problem measures the balance of an allocation as the difference between the largest site resource unit size and the smallest, a very conservative metric in that it points out the worst-case imbalance. The algorithm outlined is heuristic in nature, in part because the problem of optimally partitioning resource units, as studied in this paper, is NP-Complete, but also because the algorithm is decentralized. The consequence of the decentralized nature of the algorithm is that in performing optimizations, only a subset of the resource units in the system is considered, specifically those units on the pair of sites which currently is the most imbalanced. For this reason, the family is greedy.

Each site has associated with it a size that is the sum of the resource units sizes for the resources currently assigned to the site. Imbalance is measured as the absolute value of the difference of site sizes, a Δ. The general form of the algorithm is iterative. At each step, the pair of sites that is presently most imbalanced, i.e. the pair with the largest Δ, is selected for optimization. A balance condition defines when iteration terminates and also controls the termination of a pair-wise optimization. This balance condition defines a pair of sites as being balanced when no unit can be moved from the site with larger size to the one with smaller size without reversing the roles of the large and small sites. The general

form of the algorithm does not define a rule for selecting the resource unit to move next in a pair-wise optimization. The only stipulation is that the balance condition must not be violated.

Theoretical results establish bounds on the performance of the algorithm family. We have shown that the number of pair-wise optimizations done is O(SU) where S is the number of sites in the system and U in the number of resource units. The amount of control communication is also bounded either by O(SU), when the underlying network supports broadcasts, or by $O(S^2U)$ when communication is point-to-point. Finally, we have shown that the final allocation can have a final imbalance of at most 2*max_resource_unit_size.

Two particular algorithms in the family have been presented. Both define rules for chosing the next resource unit to move in a pair-wise optimization; both are greedy. The first, mingreedy, moves next the resource unit with smallest size currently allocated to the big site. The maxgreedy algorithm relocates next the resource unit, currently assigned to the big site, whose size most closely fits the gap between the two sites sizes without violating the balance condition.

A simulation study was conducted to measure the performance achieved with both the min and maxgreedy algorithms. We have found that:

- The form of the results on all performance metrics does not depend on the resource unit size statistics. However, the resource unit size statistics can change the values of the the final maxΔ parameter. The number of optimizations and normalized cumulative resource unit

size moved were unaffected by resource unit size distribution.

• The form of the final maxΔ, as a function of the number of units, is very dependent on the number of sites. The number of optimizations and cumulative moved size depended more on the number of sites than the number of units. In general, with more sites, more optimizations were performed. Normalized moved size was either flat in the number of units or decreasing.

• The initial distribution of resource units to sites is very important in determining the performance of the algorithms. As the initial allocation becomes more distributed, the number of optimizations decreases as do both the normalized moved size and the final maxΔ.

• We found that the mingreedy algorithm performed better in terms of number of optimizations and normalized moved size. For the final maxΔ, mingreedy was better than max only for a centralized initial allocation. For semi- and fully distributed initial allocations, the maxgreedy outperforms mingreedy by very significant amounts.

References

[1] J. Bannister, and K. Trivedi, "Task allocation in fault-tolerant distributed sytems," *Acta Informatica*, vol. 20, pp. 261-281, 1983

[2] S. Bokhari, "Dual processor scheduling with dynamic reassignment," *IEEE Trans. Software Eng.*, vol. SE-5, pp. 341-349, 1979

[3] P. Chen and J. Akoka, "Optimal design of distributed information sytems", *IEEE Trans. Comput.*, vol. C-29, pp. 1068-1080, 1980

[4] T. Chou and J. Abraham, "Load balancing in distributed systems", *IEEE Trans. Software Eng.*, vol. SE-8, pp. 401-412, 1982

[5] W. Chu, "Optimal file allocation in a multiple computer system", *IEEE Trans. Comput.*, vol. C-18, pp. 885-890, 1969

[6] K. Efe, "Heuristic models of task assignment scheduling in distributed systems", *Computer*, vol. 15, pp. 50-56, 1982

[7] M. Fisher and D. Hochbaum, "Database location in computer networks", *Jour. Assoc. Comput. Mach.*, vol. 27, pp. 718-735, 1980

[8] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, Ca: W. H. Freeman & Co., 1979

[9] D. Gusfield, "Parametric combinatorial computing and a problem of program module distribution", *Jour. Assoc. Comput. Mach.*, vol. 30, pp. 551-563, 1983

[10] K. Irani and N. Khabbaz, "A methodology for the design of communication networks and the distribution of data in distributed supercomputer systems," *IEEE Trans. Comput.*, vol. C-31, pp. 419-434, 1982

[11] L. Laning and M. Leonard, "File allocation in a distributed computer communication network", *IEEE Trans. Comput.*, vol. C-32, pp. 232-244, 1983

[12] P. Ma, E. Lee and M. Tsuchiya, "A task allocation model for distributed computing systems", *IEEE Trans. on Comput.*, vol. C-31, pp. 41-47, 1982

[13] H. Morgan and K. Levin, "Optimal program and data locations in computer networks", *Commun. Assoc. Comput. Mach.*, vol. 20, pp. 315-322, 1977

[14] C. Ramamoorthy and B. Wah, "The isomorphism of simple file allocation", *IEEE Trans. Comput.*, vol. C-32, pp. 221-232, 1983

[15] G. Rao, H. Stone and T. Hu, "Assignment of tasks in a distributed processor system with limited memory", *IEEE Trans. Comput.*, vol. C-28, pp. 291-299, 1979

[16] C. Shen and W. Tsai, "A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion", *IEEE Trans. Comput.*, vol. C-34, pp. 197-203, 1985

[17] H. Stone, "Critical load factors in two-processor distributed systems", *IEEE Trans. Software Eng.*, vol. SE-4, pp. 254-258, 1978

[18] H. Stone and S. Bokhari, "Control of distributed processes", *Computer*, vol. 11, pp. 97-106, 1978

[19] H. Stone, "Multiprocessor scheduling with the aid of network flow algorithms", *IEEE Trans. Software Eng.*, vol. SE-3, pp. 85-93, 1977

[20] Y. Wang and R. Morris, "Load sharing in distributed systems", *IEEE Trans. Comput.*, vol. C-34, pp. 204-217, 1985