A Microeconomic Approach to Optimal File Allocation [1]

JAMES F. KUROSE
RAHUL SIMHA

COINS TECHNICAL REPORT - TR 85-43

Department of Computer and Information Science
University of Massachusetts
Amherst, Mass. 01003

Abstract

A decentralized algorithm is presented for optimally distributing a file over a network in the case that the file may be split or fragmented over several nodes. The optimization criteria include both the communication cost of accessing the file as well as the average delay associated with file access.

Our iterative algorithm has its origins in the field of mathematical economics. It is shown to have several attractive properties, including its simplicity and distributed nature, the computation of feasible and increasingly better file allocations as the result of each iteration, and rapid convergence. Conditions are formally derived under which the algorithm is guaranteed to converge and the behavior of the algorithm is additionally tested and examined through simulation. The problem of extending the model to more general file allocation schemes is also addressed.

[1] This work was supported in part by the National Science Foundation under Grant DMC-8504793 and by a Faculty Development Award from the International Business Machines Corp.

1. Introduction

During the past ten years, we have witnessed a tremendous increase in the availability of and interest in distributed computer systems. In the broadest sense, these systems can be thought of simply as a set of interconnected *computing agents* which require the use of certain system *resources* in order to perform their assigned tasks. Since significant benefits can often be realized by *sharing* these system resources among the distributed agents, a principal challenge in the area of distributed system design is the development of efficient and robust resource allocation and access mechanisms.

In this paper, we consider a classical resource allocation problem, the file allocation problem (FAP) [11] [36] [8], from a somewhat novel point of view, by drawing an analogy between FAP and the general resource allocation problem in an economy. In FAP, accesses (queries and updates) to a set of files are generated by the distributed agents. The FAP problem then is to allocate the one or more copies of the one or more files among the various nodes in the distributed system in order to optimize some system-wide performance metric. The main result of this paper is an iterative, *decentralized* optimization algorithm for solving certain versions of FAP; the process of broadening these results for more general file allocation problems is also addressed. This algorithm is shown to have several attractive features including its simplicity, distributed nature, provable (and rapid) convergence, and the computation of successively better file allocations at each step of the algorithm. We believe that these properties make this algorithm particularly well-suited for actual implementation in a distributed system.

Our approach is developed using ideas and methods previously developed [15] [18] for another well-established distributed environment - an economy. In the past thirty years, mathematical economists have developed elegant normative models describing how resources in an economy may be optimally shared in informationally and computationally decentralized ways. We believe that the numerous similarities between economic systems and distributed computer systems suggest that *models* and methods previously developed within the field of mathematical economics can serve as *blueprints* for engineering similar mechanisms in distributed computer systems. Our current work is an effort supporting this belief.

In subsequent sections we will more precisely define the FAP problem and survey much of the past research in this area. As we will see, our approach differs from previous work in this area in several important respects. Perhaps most importantly, the various FAP solution techniques examined in the past have been *centralized* in nature. By contrast, the FAP algorithm developed in this paper is *decentralized* and iterative in nature, consisting at each step of a straightforward *local computation* followed by a communication step among those agents which require access to a given file. The optimization algorithm itself is based on the simple and intuitive (but also formally proven) principle that a system's overall performance level will increase if resources (in this case, files) are redistributed in such a way as to increase the allocation of resources where the *marginal*

utility (i.e., *change* in performance) associated with such a change is above average and to decrease the allocation where the marginal utility is below average.

In the following section we examine two possible approaches towards solving general resource allocation problems based on a microeconomic paradigm. In sections 3 and 4, we survey past research on FAP and then formulate our model of the problem. In section 5, we present the decentralized FAP algorithm and formally examine its properties. Our theoretical work has been augmented by extensive computer simulation of the model and some of the important results of this simulation can be found in section 5. In section 7 we discuss an extension to the model that incorporates multiple copies of files. Section 8 summarizes this paper and discusses related problems for future research. The mathematical derivations of the results discussed in section 5 are presented in the appendix of this paper.

2. Microeconomic Approaches in Distributed Systems

The normative ("what should or could be") study of resource allocation problems in mathematical economics has had a long history. In pure exchange economies, in which a fixed amount of resources may be exchanged (allocated) but no additional resources can be produced, two basic approaches towards developing decentralized resource allocation mechanisms can be identified [21]: *price-directed* and *resource-directed* approaches.

In the price-directed approach [22] [3], [19], [10], each economic agent is allocated some initial amount of the resources it requires and an arbitrary, non-negative, initial set of system-wide prices is chosen for the resources. The agents then compute their demands for resources as that allocation of resources which optimizes their *individual* (and hence "selfish") performance metric or *utility function* within certain price constraints. The prices then change to accommodate the demands and the process continues until the total demand for a resource exactly equals the total amount available; at this point each agent can then be allocated its demanded amount of each resource. One of the major results from microeconomic theory [3] states that once the above process has converged, the resulting final allocation of resources is provably *Pareto optimal*. This means that there exists no alternate distribution of resources for which the individual utility of each agent is greater than or equal to the utility resulting from the final allocation of resources as computed above. Note that the condition of Pareto optimality is considerably weaker than the traditional notion of the optimum of some system-wide (social) utility function.

This price-directed approach towards resource allocation has been used in [24] in developing an algorithm for optimizing network transmission times in multiple access networks. The use of pricing as a prioritization mechanism in computer systems was also studied in [1] and [24].

We note that the price-directed approach has several qualities which are desirable in a decentralized algorithm: each node operates individually, optimizing its own utility function. The

process is thus both computationally and informationally decentralized. There are, however, several drawbacks to this method:

- There is no guarantee that the method will result in a feasible allocation (i.e., that the amount of a resource demanded equals the amount available) except at the optimum. This means that the process must converge in order to ensure a feasible allocation.
- Successive iterations of the procedure may not result in a monotonic increase in utility. Only after convergence can we be certain of an increase in utility.
- Each agent computes its demands by solving a local constrained optimization problem (optimizing its utility), which can be in itself a non-trivial problem. Furthermore, due to the local maximization process, proving convergence properties of the algorithm is much more difficult than in the resource-directed case.
- The method works well for agents optimizing their selfish (local) utility. However, Pareto optimality is a weaker condition than system-wide optimality, and considering the social utility derived as a consequence of individual utility functions considerably complicates the price-directed process.

Our decentralized FAP algorithm to be described in section 5 belongs to the second broad class of decentralized resource allocation mechanisms, known as resource-directed approaches [15], [18], [20]. In this approach, agents again receive an initial allocation of resources (files) but begin an iterative process of local (and hence parallel) computations followed by a subsequent *immediate reallocation of resources*. During each iteration, each agent computes the *marginal* value of each resource it requires given its current allocation of resources (i.e., computes the partial derivative of its utility function (performance) with respect to that resource, evaluated at the current allocation level). Each marginal value is then broadcast to the other economic agents which require use of this resource.

The allocation of a resource is then changed such that agents with an above average marginal utility receive more of this resource and agents with a below average marginal utility are allocated less of the resource. A particularly attractive feature of this process [15], [21] is that if the initial allocation is *feasible* (the total amount of resources allocated equals the total amount available), so too are the later allocations. Moreover, successive iterations of the algorithm result in resource allocations of strictly increasing system-wide utility. These two properties of *feasibility* and *monotonicity* will be formally established in section 5. Note that together they ensure that the resource allocation process can be terminated before convergence and the resulting final allocation is guaranteed to be both feasible and also to result in a system-wide utility strictly superior to that achievable under the initial allocation of resources.

3. The File Allocation Problem

The file allocation problem (FAP) has been the topic of numerous research efforts over the past seventeen years. Since a thorough review of FAP would itself constitute a rather lengthy survey paper (see [36], [11]), in this section we will only briefly define FAP and overview the past work of others in order to place our present effort in the appropriate context. The interested reader is referred to these survey articles (in particular, the excellent survey in [11]) and the references cited below for additional details.

Simply stated, the file allocation problem addresses the question: "Given m copies of n files, how should these copies be allocated among the N nodes in a distributed system in order to "optimize" system operation?" Unfortunately, there appears to be no one universally accepted model for this problem in the literature; many different objective functions and problem constraints have been considered. However, one of two optimization goals has typically been adopted [11]: either (a) minimization of the overall communication *cost* required to satisfy the file accesses or (b) optimization of some performance-related metric such as the *average time delay* required to access a file.

When minimization of communication cost is the primary consideration, it is also often assumed that a file must reside wholly at one node, i.e., a file cannot be *fragmented* between various nodes. In this case, the optimization problem can be formulated as an integer (0/1) programming problem, as in the seminal work of Chu [8]. This particular problem formulation was later shown to be NP complete [12], thus suggesting the necessity of heuristics or approximation techniques in all but the simplest of networks. Casey [4] considered the file allocation problem with storage costs, different access (query versus update) costs and a variable number of copies of each file and demonstrated a search technique which did not require all possible allocations to be enumerated or considered. Heuristic search techniques have also been proposed and examined in [27] and [5]. In [28], the problem of optimally locating both the files and the programs that access these files, given communication and storage costs, was examined. Under the assumption that program storage (as opposed to file storage) was negligible, the multiple file cost minimization problem was shown to decompose into individual file cost minimization problems. This decomposition result was extended in [33] for the case in which link communication capacities and storage constraints were additionally considered.

When minimization of average *time delay* or maximization of *throughput* is the primary performance metric, queueing theoretic models have been adopted. In this case, the restriction that files be wholly allocated at a node has often been relaxed. As hypothesized in [30], and as actually demonstrated in the following section, permitting such file fragmentation may result in a performance improvement (both in terms of cost and time delay) over the case of strictly integral file allocations. In [29], FAP (under the restriction that a single, central node generates all file accesses)

was modeled as a memory hierarchy problem and an algorithm was presented which generated a file assignment which minimized average access time. Chen [7] generalized this work and presented a centralized algorithm which allocated files in order to minimize access delay in the case that any network node can generate a file access. We note that this latter formulation of FAP is similar to both the optimal load balancing problem [35] and the problem of optimal message routing in computer communication networks [32] [16]. Algorithms previously developed to solve these problems may thus also be applied to this formulation of the FAP problem. In particular, the work of [16] and [26], on decentralized algorithms for routing and load balancing, respectively, are closely related to the work described in the following section.

Finally, before presenting our model of the file allocation problem, we should again stress an important difference between our present effort and the FAP work surveyed above. In earlier work, almost all FAP solution techniques have been *centralized* in nature, while the algorithm presented in the following section is *decentralized* in nature. In the centralized approach, a centralized FAP optimization problem is formulated and the problem is then solved; in practice this would be done either on-line by a centralized system management function or off-line by the system operator/designer. The results of the optimization process are then imposed on the distributed agents in the system. There are several drawbacks to such a centralized approach. First, an inherent drawback in any such scheme is that of reliability since the single central agent represents a single point-of-failure in the network. A second drawback is in terms of the cost and performance of such a centralized computation. A centralized optimization requires information about the operational parameters (access statistics, constraints, priority information, etc.) and the functional form of the relevant performance metrics for *each* network agent; transmitting this information to the network manager can be a long and costly process. Furthermore, the optimization problem itself may be an extremely computationally complex task. A centralized approach towards optimization ignores the distributed computational power inherent in the network itself and instead utilizes only the computing power of the single agent performing the optimization.

4. Model Formulation

Our model of a distributed computer system is shown in figure 1. The system consists of N nodes which are interconnected through a communications network; the network is assumed to be logically fully connected in that every node can communicate (perhaps only indirectly, i.e., in a store-and-forward fashion) with every other node. The processes running at each of the nodes generate accesses (queries and updates) to the file. If a process generates a file access request which can not be satisfied locally (i.e., the portion of the file accessed is not stored locally), the access is transmitted to another node in the network which *can* satisfy the access request.

We will not distinguish here between the cost of queries and updates and will refer to each simply as an *access*. We will assume that each node, i, generates file accesses according to a poisson process

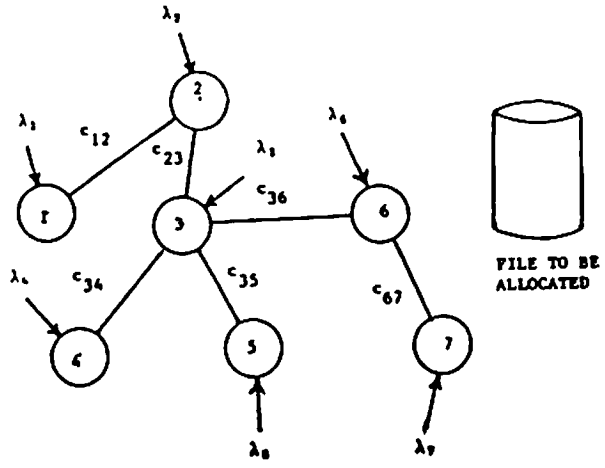

Figure 1: A distributed computer system

with parameter λ_i. In order to further simplify the presentation of our optimization procedure, we will also only consider the problem of allocating one copy of a single file among the nodes in the network. The relaxation of each of the above assumptions will be discussed in section 5.4.

As in the centralized approaches of [29] and [7], we will also assume the file is divisible in the sense that any fraction of the file can be stored at any node. In such a scenario, when a process needs to access certain records in a file, it would use some table look-up (directory) procedure in order to determine to which node it should address its file access. From a practical standpoint, we note that fragmenting a file may be beneficial for a number of reasons:

(a) Graceful degradation - If the file is distributed over a number of nodes then failure of one or more nodes only means that the portions of the file stored at those nodes cannot be accessed. File accesses are, therefore, not completely disabled by individual node failures.

(b) Performance enhancement - when two nodes simultaneously request access to portions of the file which are stored at a single node, queueing delays will be incurred, since one access must be postponed while the other access is processed. By distributing the file we can introduce parallelism into file usage (in terms of access) since different parts of the file stored in different nodes can be accessed in parallel.

Consider then a network of N nodes indexed by $i \in \{1, \ldots, N\}$, and let us define the following terms.

x_i the *fraction* of the file stored at node i. Since there is only one file in this restricted case we have $\sum_{i=1}^{N} x_i = 1$. We will assume that the individual records with a file are accessed on a uniform basis (although this can be easily relaxed) and therefore x_i also represents the probability that a file access (from anywhere in the network) will be transmitted to node i for processing.

λ_i the rate at which node i generates accesses to the file. We will assume that this process is poisson with parameter λ_i. The network-wide access generation rate is denoted by λ and

defined:

$$\lambda = \sum_{i=1}^{N} \lambda_i$$

c_{ij} the *communication* cost of transmitting a file request from node i to node j (presumably to access a portion of the file stored at j) and transmitting the response from j back to i. c_{ii} is taken to be zero.

C_i the average (system-wide) communication cost of making an access to node i. We take this simply as the weighted sum of the individual communication costs:

$$C_i = \sum_{j \in N} \frac{\lambda_j}{\lambda} c_{ji}$$

μ_i the service rate for access requests at node i. The average processing time required to satisfy a file access at node i is thus $1/\mu_i$. In order to simplify our presentation below, we will assume $\mu_i = \mu$ for all nodes i; this restriction can be easily relaxed. We will further assume that the length of service time is exponentially distributed with mean $1/\mu$. This latter assumption can be somewhat relaxed, as discussed below.

T_i the expected time delay associated with satisfying an access at node i. Note that this delay results both from the processing time at node i (the time required in order to actually access the record), as well as from queueing delays resulting from the sequential processing of file access requests at node i. Given our assumptions concerning λ, x_i and μ, we have [23]:

$$T_i = \frac{1}{\mu - \lambda x_i}$$

k a scaling factor (discussed below) that relates time delay to communication cost.

The above quantities define the individual aspects of the "costs" of accessing a file. Let us now define the system-wide cost associated with a particular allocation of file fragments to nodes. We note that this system-wide cost will consist of two components: a communication cost and a cost associated with the access time delay. Interestingly, each of these two costs considered alone, suggest diametrically opposed file allocation strategies; taken together they suggest a compromise between the two extreme strategies. Note that the communication cost term (below) is linear in x_i. Thus, if communication is the sole cost, the optimal strategy is simply to concentrate the entire file at the node where it is cheapest for the system to access (i.e., at that node, i, where C_i is minimal). On the other hand, if we consider queueing delays alone, concentrating a large amount of a file at one node means that a correspondingly large number of queries will be directed at that node with resulting large access delays. This would argue for distributing the file evenly among

the N network nodes. In our overall cost function, C, below, this tradeoff between the relative importance of communication costs and access delays is characterized by the constant k.

As discussed above, assuming the distribution of accesses to records within the file is uniform, x_i represents both the fraction of the file stored at node i as well as the probability with which a randomly chosen access is directed to the portion of the file stored at node i. Thus the overall expected cost of access to the file is given by:

$$C = \sum_{i \in N} (cost\ of\ access\ to\ x_i) prob(accessing\ x_i) = \sum_{i \in N} (C_i + kT_i)\, x_i$$

or

$$C = \sum_{i \in N} (C_i + \frac{k}{\mu - \lambda x_i}) x_i \tag{1}$$

Our optimization problem is thus to minimize C subject to the constraints $\sum_{i \in N} x_i = 1$, $x_i \geq 0$, $\forall i \in N$ and $\mu > \lambda$. This last assumption is made in order to ensure that we have finite partial derivatives. (If we do not want to restrict λ, then some functional approximation can easily be made for T_i, as in [26]). Equivalently, we can take the negative of the cost function to be our objective function and maximize this function. In order to reflect the origins of our optimization algorithm within the field of mathematical economics, we will adopt this latter problem formulation and refer to the objective function to be maximized as the *utility* of the file allocation.

$$Utility = -C = -\sum_{i \in N} (C_i + \frac{k}{\mu - \lambda x_i}) x_i \tag{2}$$

5. A Distributed Decentralized Algorithm for Optimal File Allocation

5.1 Algorithmic Approach

The decentralized file allocation algorithm presented in this section is based on a normative model of economic planning in production economies initially due to Heal [15] [18]. Heal's work provides a simple, decentralized procedure by which resources may be optimally shared among agents in an economy in which resources are both produced and consumed; our present problem concerns only the distribution of a single resource (the file) among the agents and is therefore a simplification of the more general economic planning problem.

The algorithm operates as follows. An initial allocation of file fragments is chosen by the nodes. This may be done by a single node (e.g. the node that starts the file allocation process) or may be done in a fully distributed manner according to some fixed, predetermined rules. As we will see, this initial file allocation will in no way effect the optimality of the final (computed) file allocation. It will, however, play an important role in determining the number of iterations

required by the algorithm since these initial conditions determine the "distance" between the initial and final (optimal) allocations. The only requirement of the initial allocation is that it be feasible, i.e., that $\sum_{i \in N} x_i = 1$.

Once the initial allocation is made, it is made known to the nodes in the network and the iterative process begins. Each node first computes its marginal utility (i.e., the partial derivative of the utility function with respect to x_i), evaluated at the current allocation. These N individual marginal utilities are then used to compute the *average system-wide* marginal utility. One way in which this computation can be performed is to have all nodes transmit their marginal utility to a central node which computes the average and broadcasts the results back to the individual nodes. Alternatively, each node may broadcast its marginal utility to all other nodes and then each node may compute the average marginal utility locally. (We note that in a broadcast environment, such as a local area network, these two schemes require approximately the same number of messages and thus the second scheme would be more desirable).

Upon receiving all the marginal utilities the node (or nodes) check to see if the algorithm termination criteria have been met (essentially, whether all the marginal utilities are equal). If so, the procedure stops and, as shown in the appendix to this paper and as discussed below, the resulting allocation is provably optimal. (We note as an aside that after the algorithm has converged, the resulting optimal distribution will prescribe real-valued fractions of the file to the nodes. Obviously a file cannot be divided up in this way since the divisions have to be based on the atomic elements of the file - records. It is, however, a relatively simple matter for the initiating node to start a round of communication that would round off the fractions in an appropriate way, but we will not address this issue here.)

If the termination criteria are *not* met, the file is re-allocated in such a way as to proportionally *increase* the amount of the file allocated to nodes with a marginal utility that is above average and *decrease* the amount of a file allocated to a node with a below average marginal utility. Note that intuitively, the algorithm simply serves to reallocate portions of the file to those nodes where the increase in utility (decrease in cost) will be greatest.

5.2 Algorithm Statement

A more precise statement of the algorithm is now given:

1. *Initialization*. An arbitrary allocation x_i, $i \in N$ is made.

2. Iteration. **DO**

   (a) Each node $i \in N$, calculates $\frac{\partial U}{\partial x_i}$ evaluated at the current allocation and sends $\frac{\partial U}{\partial x_i}$ and x_i to all nodes j, $j \neq i$, $j \in N$ or to the designated central agent.

(b) Each node $i \in N$, computes (initially $A = N$)

$$\Delta x_i = \alpha \left(\frac{\partial U}{\partial x_i} - \frac{1}{|A|} \sum_{j \in A} \frac{\partial U}{\partial x_j} \right) \quad , \quad \forall i \in N$$

where α is the stepsize parameter and A is a set of nodes described below.

(c) Each node $i \in A$, sets $x_i = x_i + \Delta x_i$.

UNTIL $\left| \frac{\partial U}{\partial x_i} - \frac{\partial U}{\partial x_j} \right| < \epsilon \quad \forall i, j \in A$

In the appendix, we discuss bounds on the size of α such that the above discrete file re-allocation process provably converges. As we will see, the value of α will depend on ϵ, which determines the convergence criteria of the algorithm and is typically chosen to be a very small value. Finally, the set A in the above algorithm is taken to be the set of all nodes and $|A| = N$, unless at least one node would receive a non-positive allocation under the change in file assignment. In this case, A is computed by the following procedure, which ensures that the file allocation at all nodes remains non-negative.

/* Algorithm for Computing the set A */

(i) Set $A = \{i \mid x_i + \Delta x_i > 0\}$
(ii) For all $j \notin A$, sort $\frac{\partial U}{\partial x_j}$
(iii) Select j such that j maximizes $\frac{\partial U}{\partial x_j}$, $j \notin A$
(iv) If $\frac{\partial U}{\partial x_j} > \frac{1}{|A|} \sum_{i \in A} \frac{\partial U}{\partial x_i}$ then set $A = A \cup \{j\}$
(v) Repeat steps (ii) to (iv) until no more additions can be made to A.

5.3 Algorithm Properties: Optimality, Feasibility, Monotonicity and Convergence

The algorithm presented in the previous section has several properties that make it particularly attractive for file allocation in a distributed environment. In this section we will examine and discuss these properties; the formal proofs of these properties can be found in the appendix.

The first property which we wish to demonstrate is that when the algorithm converges, (i.e., $\frac{\partial U}{\partial x_i} = \frac{\partial U}{\partial x_j} \quad \forall i, j \in A$), the necessary conditions for optimality have been satisfied.

Following [15], let $U(x_1, x_2, .., x_n)$ be the overall utility of the society of N agents. Consider the Lagrangean,

$$L = U(x_1, x_2, .., x_n) - q \cdot g(x_1, .., x_n)$$

where $g(x_1, ..x_n)$ is the constraint $\sum_{i \in N} x_i - 1$ and $g(x_1, .., x_n) = 0$. Setting

$$\frac{\partial L}{\partial x_i} = 0, \; i \in N \quad \text{and} \quad \frac{\partial L}{\partial q} = 0$$

we get

$$\frac{\partial U}{\partial x_i} = q \frac{\partial q}{\partial x_i}$$

But $\frac{\partial q}{\partial x_i} = 1$. Therefore, at optimum we have $\frac{\partial U}{\partial x_i} = q$, i.e., all the marginal utilities are equal, which is precisely the convergence criterion of our algorithm. To ensure that the additional constraint that each $x_i > 0$ we simply insure that our allocation procedure never violates the constraint. The necessary conditions for optimality are thus:

$$\frac{\partial U}{\partial x_i} = q \ \forall x_i > 0$$
$$\frac{\partial U}{\partial x_i} \leq q \ \forall x_i = 0$$

The above convergence criteria are *necessary* conditions for optimality. In general, these conditions are also satisfied by local optima, local minima, and points of inflection. For our particular problem, however, the utility function in equation 2 is convex (there are no local minima, maxima, or points of inflection) and we need only establish whether the resulting allocation results in the global maximization or minimization of utility. As we will see shortly, the above algorithm results in a *strictly monotonic* increase in utility and thus, as long as the algorithm iterates at least once, we can be assured of a globally optimum file allocation.

Another important property of the algorithm (proved in the appendix) is that it maintains a feasible file allocation ($\sum_{i \in N} x_i = 1$) at each iteration. Thus, the algorithm can be terminated prematurely (before convergence) and the resulting file allocation will be feasible (i.e., the system would be operational, although its performance would not be optimal). As discussed in section 2, this property is not found in certain other iterative decentralized optimization algorithms [24].

The third important property of the algorithm is that each successive iteration results in successively better file allocations, i.e., $\Delta U > 0$, meaning that the system-wide utility function (equation 2) is monotonically increasing as a result of each iteration, except when the conditions necessary for optimality have been satisfied. Thus, until the algorithm has converged, the cost associated with the file allocation computed as the result of the kth iteration is strictly less than the cost associated with the file allocation computed at the $(k-1)$st iteration. This property is formally established in theorem 2 in the appendix.

Note that if the algorithm is terminated before convergence, not only is the file allocation feasible but it is strictly superior to both the initial file allocation and all file allocations computed by previous iterations. This property makes the algorithm well-suited for running "in the background" (when the system nodes are free) until convergence is eventually achieved. In the meantime, the (non-optimal) intermediate file allocations computed by the algorithm can be used (with increasingly superior system performance levels) as the algorithm moves the system to an optimal file allocation.

The above two properties guarantee that once the algorithm has converged, the conditions necessary for an optimal file allocation have been satisfied and furthermore, that each computed intermediate allocation is successively better.

The final property is that for sufficiently small values of α, the algorithm *converges* to an optimal file allocation. A formal proof of convergence, initially due to Heal [15] holds for the case of infinitesimally small values of α. In the appendix, we establish an upper bound on the size of a discrete α such that the algorithm is guaranteed to converge. As shown by our simulation results, however, choosing such small values of α results in slow convergence while larger values (i.e., values *above* our bound) may result in much faster convergence. Thus, the problem of determining a less restrictive bound on α remains a challenging open question.

5.4 Generalization of the File Allocation Problem

Having formulated the file allocation problem and presented our distributed algorithm for optimally allocating a single copy of a single file over the network, let us briefly consider the problem of generalizing the preceding results to less restrictive file allocation problems. It should be clear that the optimization algorithm itself is very general in nature and can be applied to any arbitrary resource allocation problem, provided that the utility function can be properly formulated and is continuous.

Different costs for queries and updates can be easily taken into account by splitting the cost function into two separate costs (one for processing updates and one for processing queries) in equation 2 and weighting these costs appropriately. Different access processing rates at the nodes can also be trivially incorporated into the problem formulation by replacing the μ in equation 2 by the individual μ_i's. In the case of a single copy of M multiple distinct files, the utility function can be easily extended by introducing variables for additional resources, x^j, $j \in \{1 \ldots M\}$. Suppose that x_i^j is the fraction of file j allocated to node i and λ^j is the access rate to file j. Since there are now M files, the cost function of accessing a file at node i must now be summed over the M files:

$$Utility = -C = -\sum_{i \in N} \left(\sum_{j \in M} (C_i + \frac{k}{\mu - \sum_{j \in M} \lambda^j x_i^j}) x_i^j \right)$$

Note that the "cost" incurred due to time delay includes the effects of simultaneous accesses to different files stored at the same location, a real-world resource contention phenomenon which is typically not considered in most FAP formulations [11].

In the case of multiple copies of each of the one or more files, the utility function cannot be constructed simply by multiplying the individual nodal access costs by x_i. Rather, the specific contents of the various file fragments must be considered, which complicates the problem significantly.

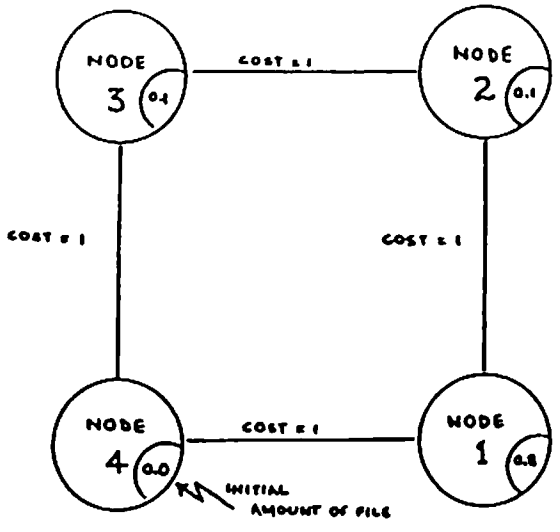

Figure 2: Network configuration

The multiple copy problem is discussed in section 7. Finally, we note that alternate queueing models (e.g., such as M/G/1 queues) can be directly used to model the access generation and service mechanisms without affecting the feasibility or monotonicity properties of the algorithm. However, the specific convergence criteria on the size of α derived in the appendix are valid only for the utility function given by equation 2 and thus similar results would have to be established for these new objective functions.

6. Experimental Results

In this section we present some of the results of computer simulations which were performed in order to test and substantiate our work. The results are in the form of graphs plotted to illustrate certain properties of the algorithm.

For the simulation we considered a four node ring network (see Figure 2) with equal link costs (note that costs of communication will therefore be identical) and the specified initial allocation. This network of marked simplicity was chosen because we knew in advance what the results should be and therefore could test the validity of the algorithm. Furthermore, tests with other configurations revealed that behavior shown by this simple four node arrangement is accurately indicative of the behavior of an arbitrary network. The routing of the access requests between any two given nodes was taken to be along the shortest (least expensive) path between the two. We took the

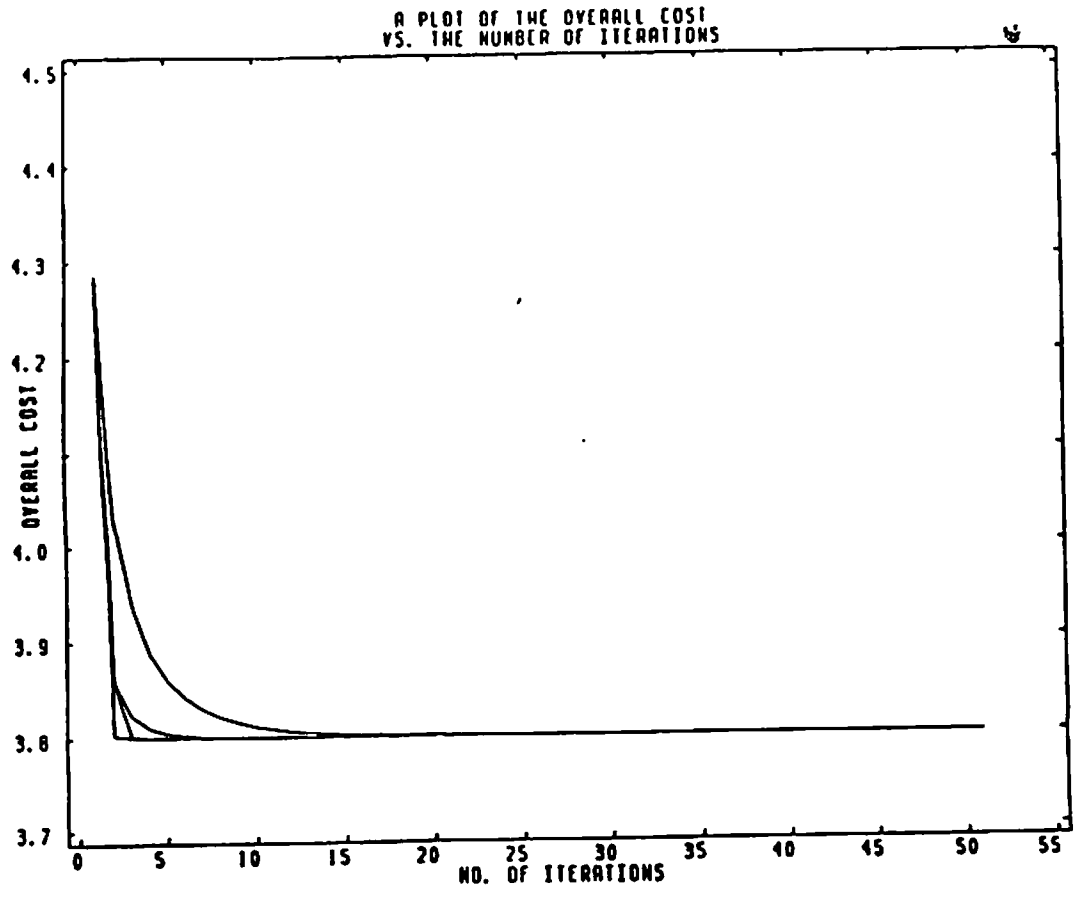

Figure 3: Convergence profiles

service rate to be 1.5 ($\mu = 1.5$), the scaling constant to be unity ($k = 1$) and the query rate to be 1 ($\lambda = 1$). Also, the stopping criterion was taken to be $\epsilon = 0.001$. This amounted to the partial derivatives being within 0.025 percent of each other when the algorithm halted.

Figure 3 demonstrates the convergence behavior of the algorithm by displaying the cost of the currently computed file allocation as a function of the number of iterations, for different values of α. Each curve represents the convergence profile for a different value of α. (4 iterations until convergence for $\alpha = 0.67$, 10 iterations for $\alpha = 0.3$, 20 iterations for $\alpha = 0.19$ and 51 iterations for $\alpha = 0.08$). The starting allocation was $(x_1, x_2, x_3, x_4) = (0.8, 0.1, 0.1, 0.0)$ and the optimal allocation, as expected, was $(0.25, 0.25, 0.25, 0.25)$. Figure 3 also demonstrates an extremely important feature of this algorithm, namely the impressive speed with which it converges. As can be seen from the graph, the overall cost (y-axis) rapidly decreases in the first few iterations (rapid convergence phase) and then gradually converges to the optimal (minimum overall cost) cost allocation. Note that even with differing values of α and final convergence times, the number of iterations in each of the rapid convergence phases are about the same.

The graph also exhibits the strict monoticity maintained by the algorithm. Thus, if so desired, ϵ can be chosen so as to restrict the number of iterations and terminate the algorithm after the rapid

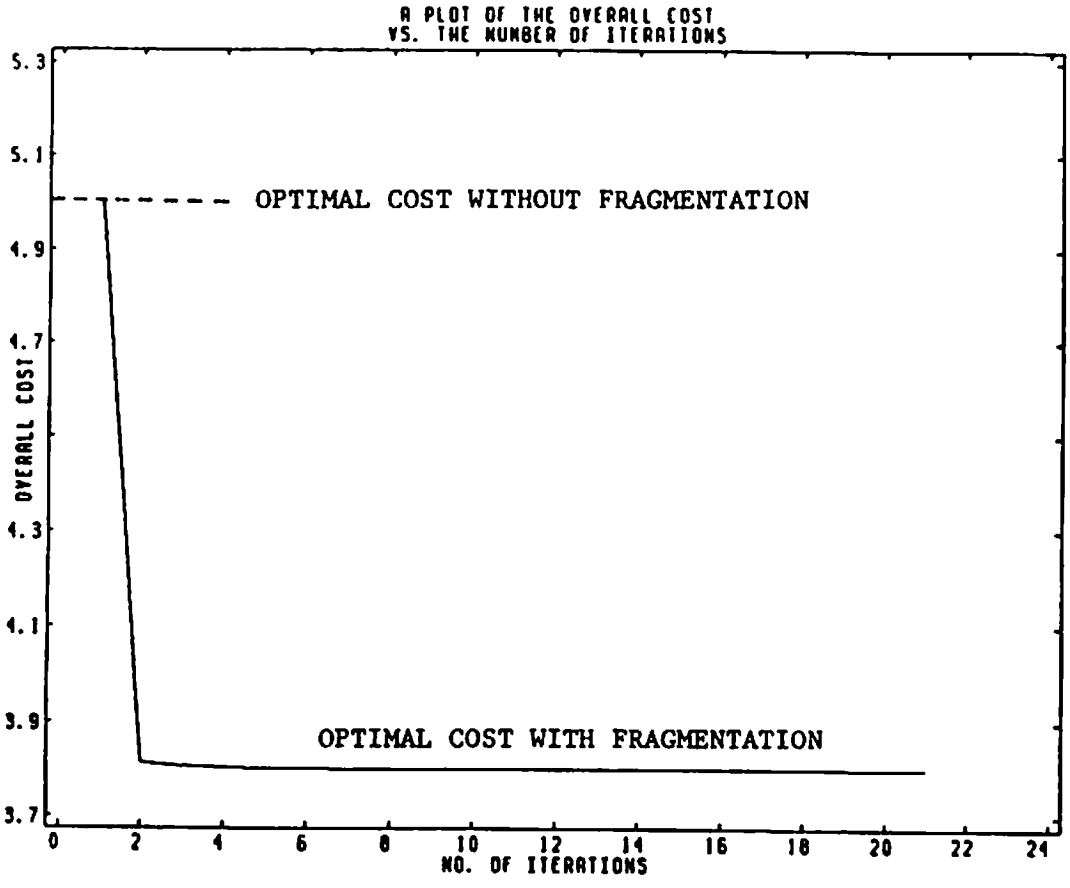

Figure 4: Starting with the entire file at one node

convergence phase. In this case, the resulting allocation would be nearly, but not quite, optimal.

Figure 4 argues in favour of fragmenting the file. The parameters μ, k and λ are as given before. The initial allocation places the entire file at one node $(0.0, 0.0, 0.0, 1.0)$ in an optimal manner given the integer allocation constraint (actually, placing the file at any node is equally optimal due to the symmetry in the network configuration). Note that the algorithm results in a significant (25%) reduction in cost at the optimal allocation $(0.25, 0.25, 0.25, 0.25)$.

In figure 5, the number of iterations required for convergence is plotted for different values of α. We see that as the values of α get smaller, convergence time increases greatly, thus making it practically very important to choose a good value for α. Note, however, that there is a relatively large range of α values which result in nearly optimal convergence speeds. Although not shown in this graph, it is also important to note that (as in figure 3) while certain α values may require a large number of iterations, it the number of iterations in the rapid convergence phase is still *relatively small.*

It is of great pragmatic interest to study the convergence behavior of the algorithm with increasing problem size. problem is increased. In figure 6 we have plotted the number of iterations needed (using the best possible α) for different network sizes. Each network of N nodes,

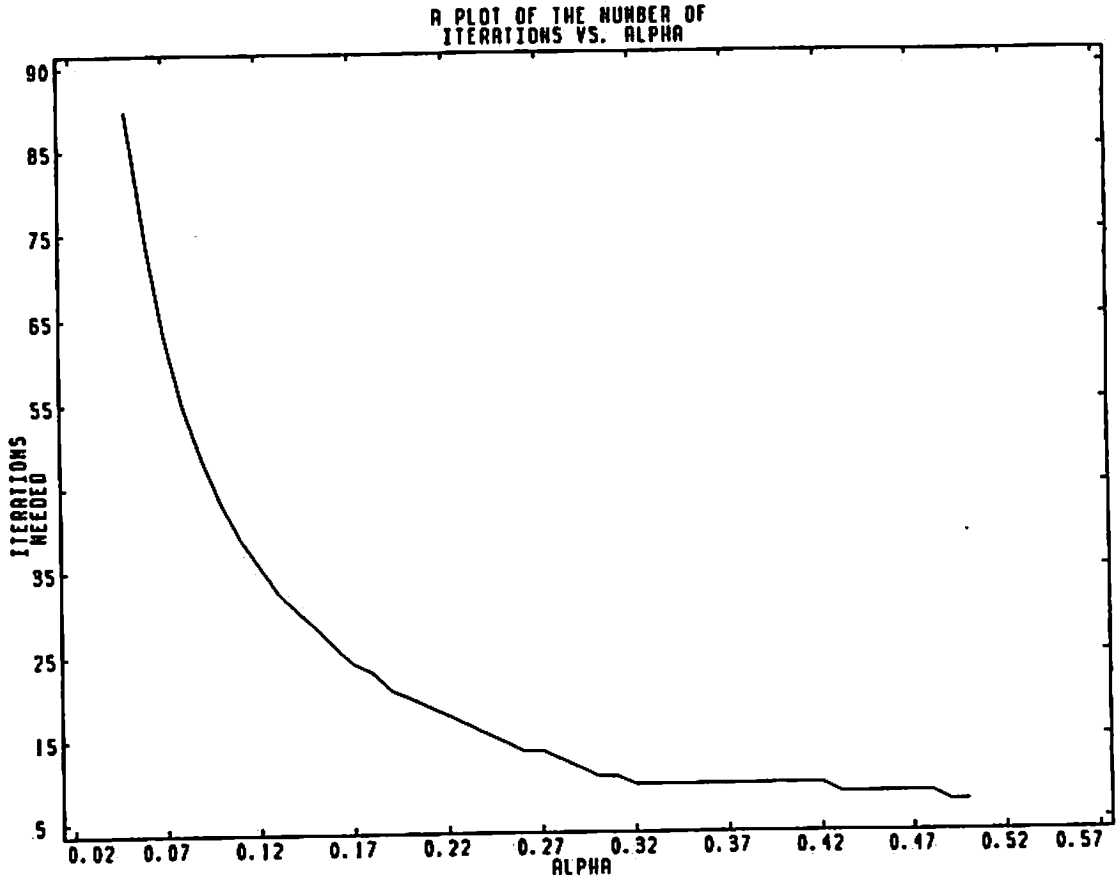

Figure 5: How a choice of α affects convergence time

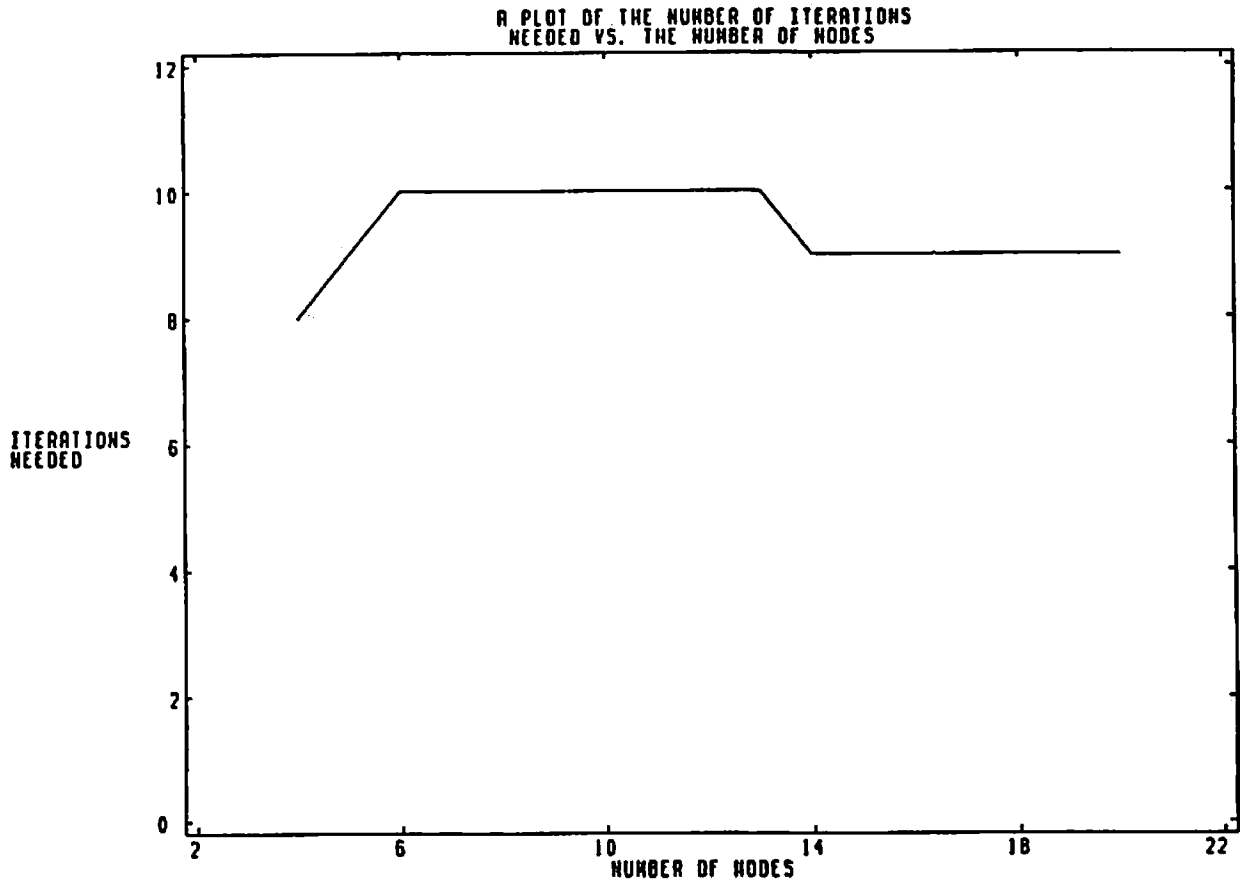

Figure 6: Increasing the number of nodes

$4 \leq N \leq 20$, is taken to be fully connected with link costs being unity. The starting allocation was $(0.8, 0.1, 0.1, 0.0, 0.0...)$ and the resulting optimal allocation was $\frac{1}{N}$ as expected. A salient feature of the algorithm, as demonstrated by this graph, is that increasing the problem size does not significantly increase the number of iterations required. Thus the algorithm's running time (in terms of the number of iterations) is not greatly affected by increasing the number of nodes. We believe that the reason for this lies in the fact that although (on increasing the number of nodes) the average marginal utility will be smaller, the size of the Δx_i and the final amounts allocated will also be smaller. That is, increasing the problem size simply scales the magnitude of the quantities involved but does not severely influence the convergence time.

7. An extension to multiple copies of files

7.1 Introduction and motivation

So far we have discussed the problem of distributing a *single* copy of a file over a number of nodes in a network. As a result of fragmenting a file there was a decrease in access costs and an increase in reliability (due to graceful degradation, as discussed in section 4) and performance (different portions of the file can be accessed in parallel). An obvious extension of this performance and reliability obtained by distribution is to permit *multiple* copies of files to reside in the network. Carefully placing different copies of files in various locations around a network will enable *parallel access*, reduce *access costs* and increase *reliability* against node failure.

This aspect of file allocation has been examined in several treatments of this problem mentioned earlier in section 3. Some of these methods solve for optimal placement of copies of files in a network. However, as pointed out before, these approaches suffer from the same drawbacks as the other integer programming approaches. The algorithms are centralized and prove to be computationally intractable with growing problem size.

Our approach then is to integrate the allocation of multiple copies into the scheme presented in the past sections and to address the problem of applying the algorithm of section 5 to optimize the allocation. Thus we consider the problem of allocating m copies of a file in a network of n nodes.

To illustrate the inherent difficulty in solving this problem let us consider a situation with multiple copies of one file in which the algorithm has converged. We may therefore expect each node in the network to contain a copy of the file or a fraction thereof. Since there are a number of copies of the file in the network, there will be many subsets of nodes, each of which as a group contain a complete copy of the file. For example, suppose that in some network configuration, nodes 3 & 4 may contain a copy of the file distributed between them and also, nodes 5 & 6 may contain a copy of the file between them. Now, node 2 has a choice of accessing two copies of the file, one at nodes 3 & 4 and the other at nodes 5 & 6. Only *one* of these may be optimal in terms of cost. Therefore each node must determine which subset of nodes that it must direct accesses to for

the portions of the file that it does not already possess. This choice of subsets must also be done to optimize access and delay costs and therefore compounds the allocation problem. Since the choice of the best subset for a node, in a given iteration, is dependent on the current allocation, the cost function itself will *change* with each allocation and considerably complicate the allocation process. For example, in the tenth iteration node 2 may find it optimal to access the copy at nodes 3 & 4 and the allocation may change so that in the eleventh iteration node 2 finds it optimal to access the copy at nodes 5 & 6. The number of such subsets from which a node can access the file can be much larger than the number of copies of the file. In the example above, if each of the nodes 3,4,5 and 6 contain 0.5 of the file (and thus adding up to two copies) then node 2 can choose from the subsets $\{3,4\},\{4,5\},\{5,6\}$ and $\{3,6\}$. For the cost to be determined at every iteration, each combination of node and subset must be evaluated for its potential contribution to the cost and the best possible combinations must be chosen in order to minimize the cost. Thus, as the number of nodes and copies of files increase, the evaluation of the cost function appears to degenerate into a combinatorial explosion. In the discussion that follows we introduce some structure into the problem to reduce the difficulty in determining the access costs at each iteration.

7.2 Multiple copies in a virtual ring network

As a first step towards dealing with multiple copies we examine the problem of allocating multiple copies of files in a *virtual ring network*. We distinguish a virtual ring network from a ring network in the following sense. A virtual ring is constructed from an arbitrary network by imposing an ordering on the nodes and establishing a protocol of communication that embeds this ordering. In other words, each node will communicate (for the purpose of file access) directly with one designated neighbour node. Thus a ring network by virtue of its structure is also a virtual ring network. The use of such rings, we believe, is sufficiently widespread to justify this simplification.

Figure 7 shows a ring of seven nodes with an allocation in terms of the fraction of the file present at each node. We have considered a unidirectional ring with copies of the file laid out *contiguously* around the ring. For example, in the above ring we may keep one copy of the file between nodes 6 and 2 and the other copy between nodes 3 and 5. Thus the first few records of the file will be kept at node 6, the next few at node 7, thereafter at nodes 1 and 2, the last few records being at node 2. Similarly, the other copy is 'stretched out' between nodes 3 and 5. Note that placing the copies of the files end to end around the ring causes the file to be contiguous at any node. There is no need to distinguish between the top and bottom of the file. Thus, node 1 sees the file starting at itself and extending upto node 4.

Now if the allocation changes so that node 2 acquires 0.8 of the file then the communication cost for node 1, in terms of the links present, decreases to only one link cost. Thus it is possible for the cost to change due to the addition or subtraction of link costs in a single iteration. Note that the marginal utilities will therefore change in jumps, the jumps being whole link costs. More abstractly,

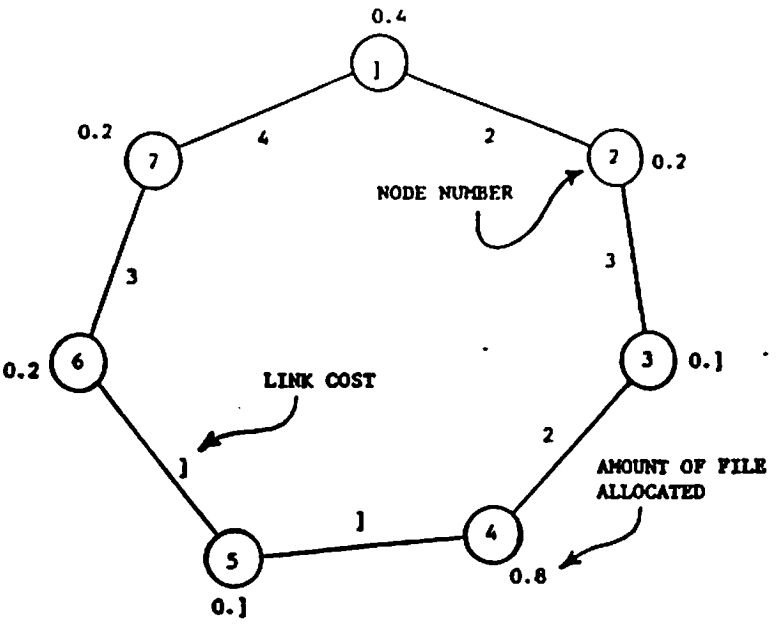

Figure 7: A virtual ring

we may say that the objective function has discontinuities and the first partial derivatives at these discontinuities are different depending on the direction of approach. As we will see later, this is the crux of the difficulty encountered in dealing with multiple copies.

Once again we define the cost function as follows:

$$C = \sum_{i \in N} C_i$$

where C_i is the cost to the system for accesses directed to node i. This cost includes both the link costs as well as delay at the node due to the access traffic directed there.

To include a full description of the cost function, which is very lengthy, would consume too much space. Due to the similarity with the cost function presented earlier in section 4 it should suffice to give an example. In the above diagram, the cost to the system of placing 0.8 of the file at node 4 would be computed as follows. The communication cost would be

$$11 * 0.1 + 7 * 0.3 + 5 * 0.7 + 2 * 0.8 + 0 * 0.8 = 8.3$$

The first term denotes the communication cost to the system of the accesses that node 7 makes to node 4. Node 7 only requires 0.1 of the file; it accesses the remaining 0.9 of the file from nodes that come earlier in the ring i.e. nodes 1, 2 & 3. The delay term is calculated using the same M/M/1 formulation described earlier with the arrival rate $\lambda = 0.1 + 0.3 + 0.7 + 0.8 + 0.8 = 2.7$. The algorithm then computes the partial derivatives $\frac{\partial C}{\partial x_i}$ and iterates for the next allocation in the same manner as before except that since there are m copies of the file we now extend the constraint $\sum_{i \in N} x_i = 1$ to $\sum_{i \in N} x_i = m$. Note that there is no restriction on limiting the number of copies

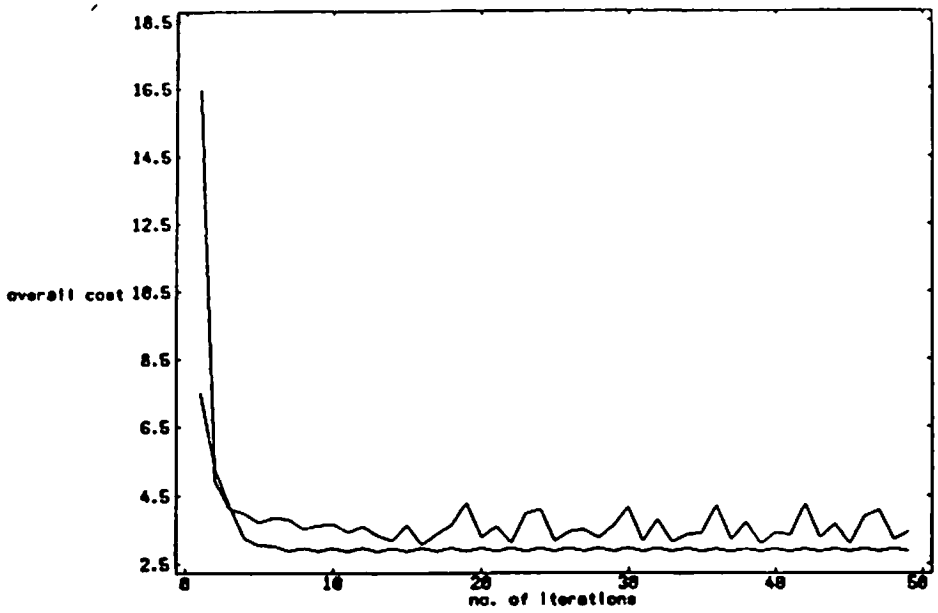

Figure 8: Convergence profiles

present at a single node i.e. that $x_i < 1$. As an example, consider a case where there are four nodes, three of which have very poor service rates i.e. the rate at which they can process an access is exceedingly slow. Therefore, it will be cheaper for the system to place most of the file at the node where service is best (given that communication costs are the same). The algorithm simply reallocates the resources within the system and has no control on the amount of resource. Thus it is possible, in the case of two copies (i.e. $m = 2$) that for the 'good-service' node above, the allocation may be 1.7 which is more than a whole copy. However, this allocation is no better than an allocation of 1.0, the whole file. But the algorithm, which can only redistribute the resource, cannot (and need not) incorporate this constraint into its execution. Thus, a node can be allocated more than a whole file, if that is what is cheaper for the system. This is not as disadvantageous as it may seem for it is a simple matter to ensure that no more than a whole file resides at a node. This can be done after the algorithm has run to completion and when the system is about to actually distribute the files.

7.3 Experimental results

Figure 8 shows two convergence profiles for a four node virtual ring with $m = 2$ files in the system. One curve, the one with greater oscillation, is the profile for the ring with link costs=(4,1,1,1). For this ring the communication cost dominates the delay term, i.e. the second term in equation 1 is dominated by the first. The other profile is for a ring with unit link costs in which the delay term dominates the communication cost. A dominant communication cost is likely to result in greater oscillation than in the case where the delay term is larger than the communication cost.

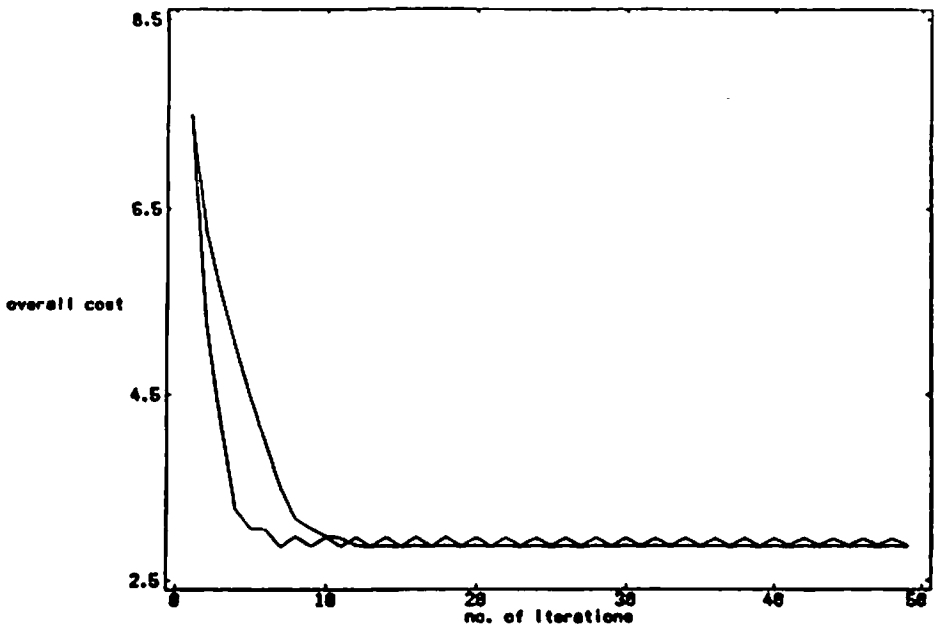

Figure 9: decreasing α

In the course of the execution of the algorithm, the communication cost term changes as links are suddenly added or subracted, depending on the allocation. As discussed earlier, this causes large and sudden changes in the partial derivatives (due to the discontinuities in the objective function) which result in large changes in the amount of a file allocated to some nodes (according to equation 1). Stated differently, the reallocation is based on the marginal utilities with the understanding that the marginal utilities should converge to being equal at optimum. Thus the abrupt changes in marginal utilities in succesive iterations cause oscillations and hence there is no convergence.

This difficulty in obtaining convergence for the above cost function is certainly a problem with any *continuous* gradient algorithm. It may appear from the discussion above that dealing with discontinuous objective functions seem to point to a serious weakness with these methods. However, we think that such a view is extreme as we may in many cases still use the algorithm and exploit its attractive features. Certainly, in situations where the delay term dominates the communications cost (which causes the discontinuities) we observe the same initial rapid phase and the later gradual phase which in this case is accompanied by small oscillations. In other cases there is still a significant decrease in cost and in these cases too the oscillations are tolerable and the algorithm can be used with a minor modification which we describe below.

Since the termination criterion of the algorithm requires that all the partial derivatives $\frac{\partial C}{\partial x_i}$ be equal, the algorithm does not converge because of the oscillations that take place around the optimum. Figure 9 shows two convergence profiles of the same ring with different values of the stepsize parameter α ($\alpha = 0.1$ and $\alpha = 0.05$). Decreasing this parameter causes the oscillations to be smaller. The termination criterion can now be modified to handle oscillatory behavior. When

oscillations are observed the value of the stepsize parameter α is decreased by a fixed amount after a certain predetermined number of iterations. When the difference in cost measured at two successive iterations is judged to be small enough the algorithm halts. We may point out that there are certainly pathological situations in which the communication cost strongly dominates the delay cost where, consequently, the oscillations are observed to be large. In this case a different halting technique has to be used such as observing the oscillations over a period of time and halting when the cost is at the lowest observed point.

Finally we note that the communication requirements of the multiple-copy version of the algorithm are greater than before since more information is needed by each individual node to calculate its marginal utility. This is the case because each node needs to know the allocation at every other node in order to be able to determine which nodes are going make an access there. Equivalently, each node could determine where it will direct its accesses to and so inform these nodes to which it will send accesses. In either case, the communication requirements are similar. In the situation prior to this section, we considered only one copy of the file and hence for any portion of the file at a given node every other node would need to make accesses there. When there are multiple copies then each node in the system must determine *who* among the nodes require to make accesses there. This extra information is needed at each iteration to compute the marginal utilities.

8. Conclusions and Future Work

In this paper, we have presented an algorithm which provides a simple, elegant and decentralized procedure for distributing a file over a network. This algorithm supports fragmentation of a file, a feature which was demonstrated to offer significant cost reduction over the case in which only integer file allocations are permitted. The algorithm was shown to have the attractive properties of maintaining feasibility, strict monotonicity and fast convergence. These properties were both formally established and studied through experimentation.

The importance of monotonicity arises in distributed operating systems where successive iterations of the algorithm can be run at freely spaced intervals (depending on load convenience), producing at each step a better allocation. One can easily envision a system where the algorithm is run occasionally at night (or whenever the system is lightly loaded) to gradually improve the allocation. The possibility also exists of using the algorithm to adaptively change the file allocation as the nodal file access characteristics change dynamically. The performance of such an adaptive scheme, however, would crucially depend on the ability of all nodes to accurately estimate the values for changing system parameters i.e. compute the partial derivatives required by the algorithm. We note that recent developments in the area of perturbation analysis [34] may provide an accurate means for estimating these partial derivatives. Finally, in the next two subsections, we correlate our file allocation model with other work done in distributed systems and outline future research in this area.

8.1 Integration with higher level abstractions

The approach we have taken in this paper is to permit the file to be fragmented across several or all of the nodes in the system. As our survey in section 3 indicates, this approach differs from much of the previous research done in the area of file allocation, in which a file resides entirely at one node and is not split across several nodes. We recognize that a great deal of research has been conducted in distributed systems and that our view of fragmenting files might pose difficulties in integrating our effort with much of this work. To this extent we examine the compatibility of our approach with certain higher level abstractions that might need to be supported by the layer of software that handles file access. Typically, this layer serves higher-level layers that require operations on data to have properties of atomicity and serializability. In this section we argue that our model of file allocation can be made consistent with the prevailing views of such abstractions.

Let us first consider the single copy instance of the file allocation problem and assess our distribution scheme with regard to maintaining atomicity and serializability. We take the view that a file is essentially a sequence of records. These records are the components of the file that reside entirely on a single node i.e. a record is not split across nodes. The algorithm we have been discussing starts with an initial allocation and, when it converges, ends up prescribing real-number fractions of the file that should be placed at each node. Obviously a file of records cannot be divided up in this manner. The real-number fractions will have to be rounded or truncated in some suitable manner so that the file, when split according to these rounded-off fractions, will fragment at record boundaries. Naturally, the larger the number of records the closer the rounded-off fractions will be to the prescribed fractions and thus the closer the final allocation will be to optimality.

We now discuss the abovementioned properties of atomicity and serializability with a view to preserving the consistency of data in the presence of concurrent transactions. Consistency is typically achieved by means of an appropriate locking scheme. If the file is fragmented over several nodes, operations that lock the whole file may require a potentially large number of messages to ensure the serializability of concurrent transactions. Thus the validity of the above argument in favour of fragmentation is premised on the assumption that most of the locking is done on the records of the file.

We observe that predicate locks [13] require an overhead in terms of messages and computation. There are two potential problems that arise with locks that hold a group of records which are spread over several nodes, one that leads to potential deadlock and the other which represents an overhead in communication. We now illustrate the two problems through a scenario.

Consider a predicate lock on ten records, five of which reside on node A with the other five on node B. Let node C send two transactions C_A and C_B that represent a single transaction to be performed on the ten records. Transaction C_A manipulates the five records at node A and transaction C_B manipulates the five records at node B. Similarly, let node D send transactions D_A

and D_B to nodes A and B. If the communication network is such that it is difficult to guarantee overall orderings of messages, then it is possible that at node A the order is C_A followed by D_A whereas at node B it is D_B followed by C_B. This would create a deadlock. To avoid deadlocks of this nature it is necessary either to make assumptions on the manner in which messages are ordered or to implement a scheme by which the predicate locks are obtained prior to any transactions being sent. The latter scheme would certainly incur an overhead in terms of the number of messages required for implementation.

The second problem arises when atomicity must be maintained. For transaction C to commit, above, it is necessary for subtransactions C_A and C_B to commit. The extra communications overhead required would not be incurred were the whole file to reside at a single node.

The above discussion seems to indicate that there are potentially serious objections to fragmenting a file over multiple nodes. We observe that these problems would constitute pathological cases of file access in systems in which predicate lock operations are dominated by other operations. Certainly, in the example above, read operations can be executed in parallel at nodes A and B. Thus, as argued earlier, fragmentation of a file introduces potential concurrency of access in the system; this may well offset any overhead incurred in supporting predicate lock operations. Thus we recognize that there exist operations for which file fragmentation may not be well-suited. However, for situations in which these operations are relatively infrequent, fragmentation is desirable because it introduces potential concurrency and reliability and in addition, permits the use of a distributed algorithm that has many desirable properties.

8.2 Future Research

We now summarize some of the possible future research in this area. and suggest extensions to our current work that would widen the scope of the theory and in addition, provide a framework for its implementation and integration into distributed systems of the future.

- Our algorithm for file allocation is based on the information about marginal utilities (first derivatives) of the participating nodes in the system. We are at the moment investigating the use of second derivative information in this algorithm. Since the reallocation is done on the basis of first derivatives, knowledge about the manner in which these derivatives are changing contributes towards a more effective algorithm. We have observed that the second derivative algorithm maintains the feasibility and monotonicity properties of the first derivative algorithm and has two additionally desirable properties. The second derivative algorithm is resilient to changes in the scale of the problem, such as would be caused by increasing the link costs or changing the service rates at the nodes. Furthermore, using second derivatives increases the tolerance of the algorithm (in terms of whether it converges or not) towards the selection of the stepsize parameter, which controls the rate of convergence of

the algorithm. Certainly, these two properties make the algorithm more resilient to changes in the parameters of the system and is therefore an improvement over the first derivative algorithm. We have already observed this behavior in a pilot study conducted earlier; however, additional work is required to prove convergence in the discrete case and to corroborate the abovementioned properties through extensive experimentation.

- Another area of study is in the topological and communications aspects of the algorithm. It is always desirable, in any distributed algorithm, to minimize the number of messages required for execution. To reduce the amount of message sending at each iteration we wish to look at restrictions in communication where nodes communicate only with their neighbours. This would certainly reduce the number of messages that are exchanged at each iteration. It would be extremely beneficial to find algorithms based on marginal utility that maintain the attractive properties of feasibility, monotonicity and rapid convergence and yet execute with a 'neighbours-only' restriction on communication. We are at present in the process of investigating two such algorithms.

- Following the study of the different algorithms mentioned above it would be worthwhile to investigate the relative rates of convergence of these algorithms. The criteria for comparison would have to include the number of iterations needed for convergence and also the number of messages passed at each iteration. If our studies indicate that the algorithms have execution times of the same order of magnitude then a comparison could be made according to the resilience to changes in parameters of the system. This would have to be done mostly by empirical observation since analysis in this area has proven to be impossibly difficult.

- In the section on multiple copies of files there are numerous issues that have not been considered. The most salient issue is: how many copies are optimal for the system? i.e. what is the best value of m? Since there are copies of files we may wish to include consistency and concurrency control costs and distinguish between reads and writes. Furthermore, the cost of storage and copy maintenance will affect the optimal number of copies. Many database systems define the resiliency of its data objects against failure in terms of the number of copies of the object. To incorporate such costs into our model of multiple copies would widen the scope of applicability of the algorithm to include such database systems. Also, the virtual ring structure may be construed as too severe a restriction to impose on an arbitrary network. It would be worthwhile to define a less restrictive topology and yet preserve the tractability of the current model. This is an indication of the amount of research that still needs to be done to arrive at a general model of multiple copies of files.

- The cost function we have taken in this paper incorporates a particular model of communication and delay costs. The value of the parameter k (see equation 1) decides the relative importance of each term in the cost function. For a practical application of the above al-

gorithm, it is important to have a rationale for choosing the value of k. Certainly, system designers require a suitable framework in which to choose values for the various parameters such as k.

- We recognize the need to look at alternative models of the cost function such as the cost function obtained by distinguishing between query and update rates. Also, it would be extremely useful to integrate file allocation with other network problems such as the classic routing problem for networks. This combined approach is based on the premise that the routing may well depend on the allocation of files itself for some networks, and it will be worthwhile to integrate the two problems and provide a joint model of cost that is to be optimized.

- In addition to the above variations in the cost model, there are other models of file access that need to be examined. For example, if we consider systems in which the whole portion of the file is copied to the querying node instead of a remote transaction working on its behalf at the destination node then the communications cost will depend on the volume of file transferred. This will indoubtedly change the cost function to include the dependency of the communication cost on the volume of file present. Such a model is useful in certain message-based distributed systems where data objects are passed by value.

- We have, in this paper, provided a theoretical value of the stepsize parameter α to guarantee convergence of the algorithm. In practice this value of α is too small to be of any real significance. It would be of tremendous interest if a theoretical model could be used to prescribe a better value of α But that is a very difficult problem and instead we might have to recourse to stepsize values suggested by the results of extensive simulation. The use and thorough testing of heuristics for the stepsize parameter α will hopefully provide a valuable guide to selecting this value for a given system.

Finally, we note that microeconomics could play an important role in the evolution of distributed operating systems. The benefits of controlling resource usage in a distributed computer system using microeconomic methods are manifold. These microeconomic methods are well-understood and established in the field of mathematical economics. Although these methods fail in certain respects to provide an exact model of real life economies, they are surprisingly well-suited to distributed computer systems. Simplicity in implementation is achieved by treating nodes as individual agents in an interacting society. We also believe that with the use of mechanisms such as pricing, intermediate and public goods, many factors affecting the usage of the resources can be easily integrated into the overall scheme. Such an approach holds great promise in that it provides a simple and decentralized framework which can significantly reduce the complexity of designing and implementing the large distributed systems of the future.

REFERENCES

[1] C. Agnew, "The Dynamic Control of Congestion Prone Systems Through Pricing", PhD Thesis, Stanford Univ., Nov. 1973.

[2] D. Bertsekas, E. Gafni, and R. Gallager, "Second Derivative Algorithms for Minimum Delay Distributed Routing in Networks, *IEEE Trans. Commun.*, Vol. COM-32, No. 12, (Aug. 1984), pp. 911-919.

[3] K. Arrow and F. Hahn, General Competitive Analysis, Holden Day Publishers, San Francisco, 1971.

[4] R. Casey, "Allocation of Copies of Files in an Information Network", *AFIPS Proc.*, Vol. 40, pp. 617-625, 1973.

[5] S. Ceri, G. Pelagatti, G. Martella, "Optimal File Allocation in a Computer Network: a Solution Based on the Knapsack Problem", *Computer Networks*, Vol. 6. pp. 345-357, 1982.

[6] S.K.Chang and A.Liu, "File Allocation in a Distributed Database", Int. Jou. of Comp. and Info. Sciences, 1982, Vol.11 No.5, pp. 325-340.

[7] P.Chen, "Optimal File Allocation in Multilevel Storage Systems", proc AFIPS 1973, Vol.42, No.2, pp. 277-282

[8] W.W. Chu, "Optimal File Allocation in a Multiple Computer System", *IEEE Trans. on Computers*, Vol. C-18, No. 10, pp. 885-889, 1969.

[9] G. Dahlquist and A. Bjork, Numerical Analysis, Prentice Hall (Englewood Cliffs, NJ), 1974.

[10] G. Debreu, Existence of Competitive Equilibrium, in Handbook of Mathematical Economics, Vol. 2, North-Holland, 1982.

[11] L. Dowdy and D. Foster, 'Comparative Models of the File Assignment Problem", *ACM Computing Surveys*, Vol. 14, No. 2 (June 1982), pp. 287-314.

[12] K. Eswaran, "Placement of Records of a File and File Allocation in a Computer Network", *IFIP Conf. Proc.*, pp. 304-307, 1974.

[13] K.Eswaran, J.N.Gray, R.A.Lorie and I.L.Traiger, "The Notions of Consistency and Predicate Locks in a Database Systems", *Comm. of the ACM* , Nov 1976 pp 624-633.

[14] D. Foster, L. Dowdy and J Ames, "File Assignment in a Computer Network", *Computer Networks*, Vol 5., pp 341-349, 1981.

[15] G. Heal, "Planning Without Prices", *Rev. of Econ. Studies*, Vol. 36, pp. 346-362, 1969.

[16] R.G.Gallager, "A Minimum Delay Routing Algorithm using Distributed Computation", IEEE Trans. on Comm., Jan. 1977 pp. 71-85.

[17] G. Heal, "Planning, Prices and Increasing Returns", *Rev. of Econ. Studies*, Vol. 38, pp. 281-294, 1971.

[18] G.M. Heal, The Theory of Economic Planning, North-Holland Publishing Co., Amsterdam, 1973.

[19] W. Hildenbrand and A. Kirman, Advanced Textbooks in Economics: Introduction to Equilibrium Analysis, North- Holland Publishing Co., Amsterdam, 1976.

[20] Y.C. Ho, L. Servi, R. Suri, "A Class of Center-Free Resource Allocation Algorithms", *Large Scale Systems*, Vol. 1, pp. 51-62, 1980.

[21] L. Hurwicz, "The Design of Mechanisms for Resource Allocation", *Amer. Econ, Rev.*, Vol. 63, No. 2, pp. 1-30, 1973.

[22] S. Karlin, Addison Wesley Series in Statistics: Mathematical Methods and Theory in Games, Programming and Economics, Addison-Wesley, Reading, Mass., 1959.

[23] L. Kleinrock, Queueing Systems: Volume 1: Theory, Wiley Interscience, New York, 1975.

[24] J.F. Kurose, M. Schwartz and Y. Yemini, "A Microeconomic Approach to Optimization of Channel Access Policies in Multiaccess Networks", 5th Int. Conf. on Dist. Comp. Sys., (May 1985, Denver, Co.), pp. 70-80.

[25] J.F. Kurose, "Time Constrained Communication In Multiple Access Networks", Doctoral Dissertation, Department of Computer Science, Columbia University, New York, 1984.

[26] J. Kurose, S. Singh "A Decentralized Algorithm for Optimum Static Load Balancing in Distributed Systems, to appear in *INFOCOM 86* (April 1986).

[27] M. Mahmoud and J. Riordan, "Optimal Allocation of Resources in Distributed Information Networks", *ACM Trans. on Database Sys.*, Vol 1., No. 1, pp. 66-78, 1976.

[28] H. Morgan and J. Levin, "Optimal Program and Data Locations in Computer Networks", Commun of ACM, Vol. 20, No. 5, 1977.

[29] C.V. Ramamoorthy and K.M. Chandy, "Optimization of Memory Hierarchies in Multiprogrammed Systems", *J.ACM,* Vol. 17, No. 3 (July 1970), pp. 426-445.

[30] J. Rothnie and N. Goodman, "A Survey of Research and Development in Distributed Database Management", *Proc. 1977 Conf. on Very Large Databases*, 1977.

[31] A. Tanenbaum, Computer Networks, Prentice-Hall, Englewood Cliffs, N.J., 1981.

[32] M. Schwartz and T. Stern, "Routing Techniques Used in Computer Communication Networks", *IEEE Trans. on Communications*, Vol. COM-28, No. 4 (April, 1980), pp. 539-553.

[33] R. Suri, "A Decentralized Approach to Optimal File Allocation in Computer Networks", *18th IEEE Conf. on Decision and Control*, pp. 141-146, IEEE Press, 1979

[34] R. Suri, "Infinitesimal Pertubation Analysis of Discrete Event Dynamic Systems: A General Theory", Proc 20th IEEE Conf. on Decision and Control", IEEE Press, 1983.

[35] A. Tantawi and D. Towsley, "Optimal Static Load Balancing in Distributed Computer Systems", *JACM* Vol. 32 No. 2 (April 1985), pp. 445-465.

[36] B. Wah, "File Placement in Distributed Computer Systems", *IEEE Computer*, Vol. 17, No. 1, (Jan. 1984), pp. 23-33.

9. Appendix: Mathematical Derivations

In formally proving the properties of the decentralized file allocation algorithm, we will find the following lemma, initially due to Heal [15], to be quite useful. We state this lemma and provide the proof here because the result is not immediately obvious and because it is central to the subsequent proofs.

LEMMA 1: For a set of real numbers a_i, $i \in N$,

$$\sum_{i \in N} a_i \left(a_i - \frac{1}{n} \sum_{i \in N} a_i \right) \geq 0$$

with equality only when $a_i = a_j \; \forall i, j \in N$.

PROOF: Let

$$avg = \frac{1}{n} \sum_{i \in N} a_i$$

Then

$$\begin{aligned}
\sum_{i \in N} a_i (a_i - avg) &= \sum_{i \in N} a_i^2 - avg \sum_{i \in N} a_i \\
&= \sum_{i \in N} a_i^2 - n \cdot avg^2 \\
&= \sum_{i \in N} a_i^2 - 2n \cdot avg^2 + n.avg^2 \\
&= \sum_{i \in N} a_i^2 - 2 \cdot avg \sum_{i \in N} a_i + \sum_{i \in N} avg^2 \\
&= \sum_{i \in N} (a_i^2 - 2 \cdot avg \cdot a_i + avg^2) \\
&= \sum_{i \in N} (a_i - avg)^2
\end{aligned}$$

which is greater than or equal to zero with equality only when $a_i = a_j$, $\forall i, j \in N$.

Our first theorem establishes the fact that a feasible file allocation is maintained at each step in the algorithm.

THEOREM 1: If the initial allocation of the file is feasible , i.e.$\sum_{i \in N} x_i = 1$, then each iteration of the algorithm results in a feasible distribution.

PROOF: We have

$$\begin{aligned}
\Delta x_i &= \alpha \left(\frac{\partial U}{\partial x_i} - \frac{1}{|A|} \sum_{j \in A} \frac{\partial U}{\partial x_j} \right) \quad \forall i \in A \\
&= 0 \qquad\qquad\qquad\qquad\qquad\qquad\qquad otherwise
\end{aligned}$$

Therefore, if $x_i^{(k)}$ represents the allocation at node i at the k^{th} iteration and (by the inductive hypothesis) is feasible, then at the $(k+1)^{th}$ iteration,

$$\begin{aligned}
\sum_{i \in N} x_i^{(k+1)} &= \sum_{i \in N} \left(x_i^{(k)} + \Delta x_i^{(k)} \right) \\
&= \sum_{i \in N} x_i^{(k)} + \sum_{i \in N} \Delta x_i^{(k)} \\
&= 1 + \sum_{i \in A} \alpha \left(\frac{\partial U^{(k)}}{\partial x_i} - \frac{1}{|A|} \sum_{j \in A} \frac{\partial U^{(k)}}{\partial x_j} \right) \\
&= 1 + \alpha \left(\sum_{i \in A} \frac{\partial U^{(k)}}{\partial x_i} - \sum_{j \in A} \frac{\partial U^{(k)}}{\partial x_j} \right) \\
&= 1
\end{aligned}$$

In the following theorem, we derive an upper bound on the size of α in order to guarantee that the algorithm results in a monotonic increase in utility at each step.

THEOREM 2: There exists a value of α, depending on the parameters of the problem, for which each iteration results in strict monotonicity except when the algorithm termination conditions hold. (Recall that in section 5.3, it was previously shown that these conditions hold only at the optimal file allocation).

PROOF: In this proof, we take a second order Taylor series approximation to the change in utility; we will shortly derive the conditions under which it is safe to limit ourselves to a second order approximation. Our goal is to determine under which conditions the overall change in the system-wide utility function (equation 2) will be strictly greater than zero:

$$\Delta U = \sum_{i \in N} \Delta U_i > 0 \tag{3}$$

For the given utility function we see that the cross partial derivatives $\frac{\partial^2 U}{\partial x_i \partial x_j} = 0$ thus simplifying the Taylor approximation to

$$\Delta U = \sum_{i \in N} \frac{\partial U}{\partial x_i} \Delta x_i + \frac{1}{2} \sum_{i \in N} \frac{\partial^2 U}{\partial x_i^2} \Delta x_i^2 > 0$$

Substituting for Δx_i, we get:

$$\sum_{i \in A} \frac{\partial U}{\partial x_i} \alpha \left(\frac{\partial U}{\partial x_i} - \frac{1}{|A|} \sum_{j \in A} \frac{\partial U}{\partial x_j} \right) + \frac{1}{2} \sum_{i \in A} \alpha^2 \frac{\partial^2 U}{\partial x_i^2} \left(\frac{\partial U}{\partial x_i} - \frac{1}{|A|} \sum_{k \in A} \frac{\partial U}{\partial x_k} \right)^2 > 0$$

or, after rearranging, we require that:

$$0 > \alpha > - \frac{2 \sum_{i \in A} \frac{\partial U}{\partial x_i} \left(\frac{\partial U}{\partial x_i} - \frac{1}{|A|} \sum_{j \in A} \frac{\partial U}{\partial x_j} \right)}{\sum_{i \in A} \frac{\partial^2 U}{\partial x_i^2} \left(\frac{\partial U}{\partial x_i} - \frac{1}{|A|} \sum_{k \in A} \frac{\partial U}{\partial x_k} \right)^2} \tag{4}$$

The fact that α is negative reflects the necessary change in order of terms in the computation of Δx_i. Thus, the order of the two terms within brackets in part(b) of the algorithm statement (see section 5.2) will be reversed so that the marginal utility is subtracted from the average. This is a consequence of the utility function being the negative of the cost function. We need only consider the magnitude of α and therefore we rewrite the above condition to yield:

$$\alpha < \frac{2 \sum_{i \in A} \frac{\partial U}{\partial x_i} \left(\frac{\partial U}{\partial x_i} - \frac{1}{|A|} \sum_{j \in A} \frac{\partial U}{\partial x_j} \right)}{\sum_{i \in A} \frac{\partial^2 U}{\partial x_i^2} \left(\frac{\partial U}{\partial x_i} - \frac{1}{|A|} \sum_{k \in A} \frac{\partial U}{\partial x_k} \right)^2} \tag{5}$$

We now want to determine the conditions under which equation 5, and thus equation 3, hold. We therefore seek a *lower* bound on the numerator term equation 5:

$$\sum_{i \in A} \frac{\partial U}{\partial x_i} \left(\frac{\partial U}{\partial x_i} - \frac{1}{|A|} \sum_{j \in A} \frac{\partial U}{\partial x_j} \right)$$

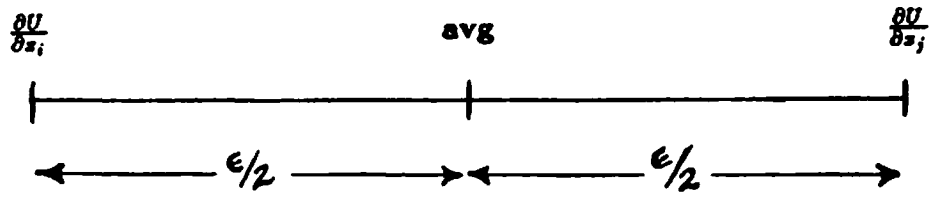

Figure 10: A worst case distribution of marginal utilities

and an *upper* bound on the denominator term:

$$\sum_{i \in A} \frac{\partial^2 U}{\partial x_i^2} \left(\frac{\partial U}{\partial x_i} - \frac{1}{|A|} \sum_{k \in A} \frac{\partial U}{\partial x_k} \right)^2$$

(i). Bounding the numerator

First let us derive a lower bound on the numerator . The stopping criterion for the algorithm (i.e. the optimality condition) is when

$$\left| \frac{\partial U}{\partial x_i} - \frac{\partial U}{\partial x_j} \right| < \epsilon \quad \forall i, j \in A, i \neq j$$

If the current allocation is not optimal then there exist $i, j \in A$ such that

$$\left| \frac{\partial U}{\partial x_i} - \frac{\partial U}{\partial x_j} \right| > \epsilon$$

In the case which minimizes the numerator, all $\frac{\partial U}{\partial x_k}, k \in A, k \neq i \neq j$, may lie between $\frac{\partial U}{\partial x_i}$ and $\frac{\partial U}{\partial x_j}$ as shown in figure 10. Therefore the average must lie in between. We know from Lemma 1 that

$$\sum_{i \in A} \frac{\partial U}{\partial x_i} \left(\frac{\partial U}{\partial x_i} - \frac{1}{|A|} \sum_{j \in A} \frac{\partial U}{\partial x_j} \right) = \sum_{i \in A} \left(\frac{\partial U}{\partial x_i} - \frac{1}{|A|} \sum_{j \in A} \frac{\partial U}{\partial x_j} \right)^2$$

It can be easily verified that the expression on the right hand side of the above equation is minimized when the average and all but these two derivatives are in the middle i.e. at $\frac{\epsilon}{2}$. Therefore,

$$\sum_{i \in A} \left(\frac{\partial U}{\partial x_i} - \frac{1}{|A|} \sum_{j \in A} \frac{\partial U}{\partial x_j} \right)^2 \geq \frac{\epsilon^2}{2}$$

and thus we get the following bound on the numerator:

$$\sum_{i \in A} \frac{\partial U}{\partial x_i} \left(\frac{\partial U}{\partial x_i} - \frac{1}{|A|} \sum_{j \in A} \frac{\partial U}{\partial x_j} \right) \geq \frac{\epsilon^2}{2}$$

(ii). Bounding the denominator In order to bound the denominator we use the following easily derivable facts:

$$(a)\ \ \frac{\partial U}{\partial x_i} = -\frac{\partial C}{\partial x_i}$$

$$(b)\ \ Upperbound\ on\ \frac{\partial C}{\partial x_i} = max\,(C_i) + \frac{\mu k}{(\mu-\lambda)^2}$$

$$(c)\ \ Lowerbound\ on\ \frac{\partial C}{\partial x_i} = min\,(C_i) + \frac{k}{\mu}$$

$$(d)\ \ Upperbound\ on\ \frac{\partial^2 C}{\partial x_i^2} = \frac{2\mu k\lambda}{(\mu-\lambda)^3}$$

We can now easily find an upper bound for the denominator by taking the those values for the various terms in the denominator which will maximize the denominator. We thus get

$$\sum_{i,j\in A} \frac{\partial^2 C}{\partial x_i^2}\left(\frac{1}{|A|}\sum_{k\in A}\frac{\partial C}{\partial x_k} - \frac{\partial C}{\partial x_i}\right)^2 \leq \frac{2n\mu k\lambda}{(\mu-\lambda)^3}\left((C_{max} - C_{min}) + \frac{\lambda k(2\mu-\lambda)}{\mu(\mu-\lambda)}\right)^2$$

where $C_{max} = max(C_i)$ and $C_{min} = min(C_i), i \in N$.

Taking the lower and upper bounds derived above and substituting them into equation 5, we find that $\Delta U > 0$ provided that

$$\alpha < \frac{\epsilon^2(\mu-\lambda)^4}{2nk\lambda((C_{max}-C_{min})\mu(\mu-\lambda)+\lambda k(2\mu-\lambda))^2}$$

Finally, we note that this upper bound on α may be overly restrictive, i.e., that convergence may still be achieved for larger values of α Indeed, our experimental results in section 5 demonstrated that much faster convergence can, in fact, be achieved with larger α values. We also note that we could get a better value for α if we dynamically calculate it at each iteration using the current allocation and evaluating the numerator and denominator accordingly.

Theorem 3 demonstrates that the second order Taylor series expansion of the utility function used in theorem 2 is sufficient to guarantee the monotonicity property.

THEOREM 3: A second order expansion of ΔU is sufficient to guarantee monotonicity.

PROOF: The complete Taylor expansion would include 3^{rd} ,4^{th} ..etc order terms , i.e.

$$\Delta U = \sum_{i\in N}\frac{\partial U}{\partial x_i}\Delta x_i + \frac{1}{2!}\sum_{i\in N}\frac{\partial^2 U}{\partial x_i^2}(\Delta x_i)^2 + \frac{1}{3!}\sum_{i\in N}\frac{\partial^3 U}{\partial x_i^3}(\Delta x_i)^3 + \ldots.$$

Note that the above expansion contains no cross terms like $\frac{\partial^2 U}{\partial x_i \partial x_j}$ because of the nature of our utility function. The only way in which the 3^{rd}, 4^{th} etc.. order derivatives are to cause the change in utility to be non-positive (as opposed to the positive change established by a second order expansion), is if their sum is negative and larger in magnitude than the sum of the first two terms. We also note that the partial derivatives are all of the same sign and thus, we need only consider the magnitudes of the partial derivatives below.

Let us first observe that Δx_i will be either positive or negative for different values of i. Clearly, only odd powers of negative values of Δx_i can cause ΔU to be non-positive. To consider the

effect of the negative terms, we will separate the Taylor series expansion into two parts, one containing terms that have negative Δx_i's and one containing terms of positive Δx_i. Let $B = \{i \mid \Delta x_i \geq 0\}$ and $C = \{i \mid \Delta x_i < 0\}$. Then

$$\begin{aligned}\Delta U = & \sum_{i \in A} \frac{\partial U}{\partial x_i} \Delta x_i \\ & + \frac{1}{2!} \sum_{i \in B} \frac{\partial^2 U}{\partial x_i^2} (\Delta x_i)^2 \quad + \frac{1}{3!} \sum_{i \in B} \frac{\partial^3 U}{\partial x_i^3} (\Delta x_i)^3 \\ & + \frac{1}{2!} \sum_{i \in C} \frac{\partial^2 U}{\partial x_i^2} (\Delta x_i)^2 \quad + \frac{1}{3!} \sum_{i \in C} \frac{\partial^3 U}{\partial x_i^3} (\Delta x_i)^3 + \ldots.\end{aligned}$$

We can write

$$\frac{\partial^n U}{\partial x_i^n} = \frac{n!}{2} \left(\frac{\lambda}{\mu - \lambda x_i} \right)^{n-2} \frac{\partial^2 U}{\partial x_i^2} \quad , \forall n = 2, 3, \ldots$$

Note that all the terms for $i \in B$ are positive. The above sum of terms for $i \in C$ becomes an alternating geometric series with ratio (for each i) $\frac{\lambda \Delta x_i}{\mu - \lambda x_i}$. We can now use the well established result (e.g. Theorem 3.1.4. in [9]) that the sign of the sum of an alternating geometric series will be the sign of the first term (which is positive in this case) and thus the series for $i \in C$ is positive. The sum of terms for $i \in B$ is positive and the series $i \in A$ is positive by lemma 1. Thus $\Delta U > 0$ and thus a second order expansion is sufficient.

In the derivation above, however, we required the restriction that the geometric ratio above must be less then unity, i.e.

$$\frac{\lambda \Delta x_i}{\mu - \lambda x_i} < 1$$

or

$$\frac{\mu}{\lambda} > (x + \Delta x_i)$$

But $\mu > \lambda$ and $x + \Delta x_i \leq 1$ and therefore this restriction is already satisfied. Finally, we note that the factorial coefficients in the Taylor series expansion (which we have ignored) will contribute to a further reduction in magnitude of each term. Our goal here has not been to derive precise restrictions on the problem parameters but simply rather to demonstrate the validity of a second order expansion. The main point of the theorem is to show that the evaluation of the second order expansion in theorem 2 is sufficient to insure that the value of α in theorem 2 will, in fact, guarantee that $\Delta U > 0$.

Finally, theorem 4 establishes the fact that each increase in utility is always bounded below by some finite value. This theorem, together with theorem 2 (which states that the increase in utility is always non-negative) insures that the algorithm will not produce an infinite sequence of allocations which result in a sequence of changes in utility which become increasingly smaller and smaller. This property together with the fact that the utility of an optimal allocation is non-infinite, guarantees that the file allocation algorithm eventually converges.

THEOREM 4: The second order expansion of ΔU is bounded below given a fixed ϵ and α.

PROOF: We have, as before,

$$\Delta U = \sum_{i \in N} \frac{\partial U}{\partial x_i} \Delta x_i + \frac{1}{2!} \sum_{i \in N} \frac{\partial^2 U}{\partial x_i^2} (\Delta x_i)^2$$

Since we are interested only in the magnitude of ΔU and since the sign of $\frac{\partial U}{\partial x_i}$ and $\frac{\partial^2 U}{\partial x_i^2}$ are the same we know that, using the results of Theorem 2,

$$\Delta U \geq \alpha\frac{\epsilon^2}{2} + \frac{1}{2!}\sum_{i\in N}\frac{\partial^2 U}{\partial x_i^2}(\Delta x_i)^2$$

which bounds ΔU below.