A Microeconomic Approach to Optimal Resource Allocation in Distributed Computer Systems

JAMES F. KUROSE, MEMBER, IEEE, AND RAHUL SIMHA

Abstract—**Decentralized algorithms are examined for optimally distributing a divisible resource in a distributed computer system. In order to study this problem in a specific context, we consider the problem of optimal file allocation. In this case, the optimization criteria include both the communication cost and average processing delay associated with a file access.**

Our algorithms have their origins in the field of mathematical economics. They are shown to have several attractive properties, including their simplicity and distributed nature, the computation of feasible and increasingly better resource allocations as the result of each iteration, and in the case of file allocation, rapid convergence. Conditions are formally derived under which the algorithms are guaranteed to converge and their convergence behavior is additionally examined through simulation.

Index Terms—**Distributed algorithms, distributed systems, file allocation, resource allocation, optimization**

I. INTRODUCTION

IN THE broadest sense, a distributed computer system 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 *decentralized* algorithms for solving resource allocation problems in distributed computer systems. In order to study this problem within a specific context, we consider, as an example, the classical resource allocation problem of file allocation (FAP) [10], [25], [9]. We are particularly interested in studying *distributed* resource allocation algorithms for several reasons. First, an inherent drawback in any nondistributed scheme is that of reliability, since a single central agent represents a critical point-of-failure in the network. Second, 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 a single central agent. Third, a decentralized approach is more appropriate in a network in homogeneous processors, each processor interacts with others as peers and the computational burden of resource allocation and access is equitably distributed among the processors. Finally, and most importantly, the information required at each step in an optimization process may itself be distributed throughout the system. Rather than transmistting this information to a central site at each iteration, the nodes may exchange this information among themselves and possibly reduce the communication requirements of the algorithm by exploiting the structure of the communication system or structure inherent in the problem itself.

In this paper, we examine distributed, gradient-based approaches toward resource allocation; the particular algorithms we study are based on a normative model of resource allocation from the field of mathematical economics [14]. These algorithms are shown to have several attractive features including their simplicity, distributed nature, provable (and rapid) convergence, and the computation of successively better resource allocations at each step. The main results of this paper are an empirical study and quantitative comparison of the convergence of various distributed gradient-based approaches to resource allocation, the derivation of bounds on stepsize parameters required by discrete versions of these algorithms for FAP, and a study of how the communication structure of a distributed system may be effectively exploited to expedite the optimization process. Resource allocation *algorithms* based on the economic *models* of [14] were first examined in [15]. As in [15], we are interested in decentralized algorithms. Our work differs from [15] primarily in our focus on second derivative algorithms, empirical studies of the convergence properties, generalized topology-based communication structures, and application to the file allocation problem.

In this paper, *analytic formulas* will be used to compute the system performance realized by a particular resource allocation; i.e., it is assumed that there is an underlying performance *model* of the system which is adequate for the purposes of resource allocation, an approach which has been adopted in existing systems (e.g., [4]). This information might alternatively be gained through actual observations and these *measurements* can then be used in the optimization process (in which case, we note, the information is naturally distributed throughout the system). In such cases, problems of estimation and the use of inexact information in distributed *stochastic approximation* (optimization) [26] algorithms must also be addressed. We feel, however, that it is important to first fully understand the problems posed by distributed resource allocation algorithms in the absence of these complicating factors.

The remainder of this paper is structured as follows. In the

Manuscript received September 1, 1986; revised February 10, 1987. 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.

The authors are with the Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01003.

IEEE Log Number 8825680.

following section, we overview past work on distributed resource allocation and FAP. In Section III, we then precisely define our model of FAP. The main results of the paper are presented in Section IV, in which three decentralized resource allocation algorithms are presented and studied. Section V discusses issues related to implementing these algorithms in distributed systems and Section VI summarizes this paper. The mathematical derivations of the results in this paper are presented in the Appendix.

II. Distributed Resource Allocation Algorithms and File Allocation

The distributed resource allocation algorithms we study in this paper are based on ideas and methods previously developed [14] for another well-established distributed environment—an economy. In the past 30 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 and [1], [15], [18], [23] are efforts supporting this belief.

Two basic microeconomic approaches towards developing decentralized resource allocation mechanisms can be identified [16]: *price-directed* and *resource-directed* approaches. In the price-directed approach [2], an initial allocation of resources is made and an arbitrary set of systemwide initial resource prices is chosen. Prices are then iteratively changed to accommodate the "demands" for resources until the total demand for a resource exactly equals the total amount available, at which point the resulting final allocation of resources is provably *Pareto optimal* [2]. We note that there are several drawbacks in adopting this method in a distributed system, including the fact that the pricing process must converge before resources can be allocated, a nontrivial constrained optimization problem must be solved by each economic agent at each iteration, and finally, only a weakly (Pareto) optimal allocation of resources is obtained.

The decentralized algorithms we examine belong to the class of resource-directed approaches [14], [15]. 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). These marginal values are then sent to other agents requiring use of this resource. The "allocation" of the 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. We note that when analytic formulas are used to compute the performance realized by a given resource allocation, an actual reallocation need not (but may) take place immediately after each iteration; an agent may simply compute its new allocation at each iteration and the resources may then be allocated whenever the algorithm is terminated. In the case that actual measurements are used, however, resources must be immediately reallocated in order for each agent to measure its performance under the new allocation.

A particularly attractive feature of this process 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, when analytic formulas are used to compute performance, successive iterations of the algorithm result in resource allocations of strictly increasing systemwide utility. These two properties of *feasibility* and *monotonicity* will be formally established for FAP in Section IV. These features make the algorithm well-suited for running "in the background" (when the system nodes would otherwise be idle) until convergence is eventually achieved. In the meantime, the (nonoptimal) allocations computed by the algorithm can be used (with increasingly superior system performance levels) as the algorithm moves the system towards an optimal allocation. Finally, we note that the resource-directed algorithms studied in this paper belong to the more general class of distributed, gradient-based optimization algorithms, and that other related algorithms in this class have been used to solve routing and flow control problems [13], [3], [24] in computer communication networks; we compare these approaches to those presented in this paper in Section IV.

In order to study the resource allocation problem within a specific context, we consider, as an example, the problem of optimal file allocation (FAP). In FAP, accesses (queries and updates) to a file or file system are generated by the distributed agents. Simply stated, the file allocation problem addresses the question of how the file or file system should be allocated among the nodes in a distributed system in order to "optimize" system performance. The term "file allocation" has been used to refer to both the problem of distributing a single file (or copies of a file) over the network [9], [11], [25] (in which case the unit of allocation is a file record, assuming a file may be fragmented) as well as the problem of distributing *an entire file system* over the network [27] (in which case the unit of allocation is a file). In either case, however, there is a divisible resource (the file or file system), the allocation of which will cause a certain pattern of accesses to be directed to the nodes to which the file or file system fragments have been allocated. The algorithms examined in this paper are applicable to the general problem of resource allocation and thus to both formulations of the "file allocation" problem. In the remainder of this paper, we thus refer to the resource being allocated (the file or file system) simply as a "*file resource.*"

The file allocation problem has been the topic of numerous research efforts and a thorough review can be found in [25] and [10]; here, we only briefly overview this past work. One of two optimization goals has typically been adopted [10]: either 1) minimization of the overall communication *cost* required to satisfy file accesses or 2) optimization of some performance-related metric such as the *average time delay* associated with file access. When minimization of communication cost is the primary consideration, it is also often assumed that a file resource must reside wholly at one node, i.e., it is not divisible and thus cannot be *fragmented* between

various nodes. In this case, FAP can be formulated as an NP-complete integer (0/1) programming problem [9], [11], for which heuristics or approximation techniques were investigated in [6], [20], and [7].

When minimization of average *time delay* or maximization of *throughput* is the primary performance metric, queueing-theoretic models have been adopted and the restriction that a file resource be wholly allocated at a node is relaxed [8], [21], [27]. In practice, a process would thus need to use some table lookup (directory) procedure to determine the node to which it should address a particular file access. As hypothesized in [22], and as demonstrated in this paper, performance is improved over the integer allocation case by permitting concurrent access of a file resource, since different fragments (stored in different nodes) can be accessed in parallel. Fragmentation additionally provides for increased reliability and graceful degradation since failure of one or more nodes only means that the portions of the file resource stored at those nodes cannot be accessed. File accesses are, therefore, not completely disabled by individual node failures. Centralized queueing-theoretic FAP algorithms have been examined in [8] and [21].

III. Model Formulation

Our model of a distributed computer system is shown in Fig. 1. The system consists of n nodes interconnected through a communications network; the network is assumed to be logically fully connected in that every node can communicate (perhaps only 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 resource. If a process generates an access request which cannot be satisfied locally (i.e., the information accessed is not stored locally), the access is transmitted to another node in the network which *can* satisfy the request. This requires that each node store the file fragment locations in its local lookup table and hence results in larger table sizes over the strictly integer allocation case. Also, as discussed in Section II, resources may be repositioned in the network at each iteration of the allocation algorithm, in which case the table entries must be updated to record the current location of resources in the system. If, however, the reallocation is performed only upon algorithm termination, the tables need only be updated once.

To simplify our presentation, we will initially consider the problem of allocating one copy of a single file resource. Consider then a network of n nodes, $N = \{1, \cdots, n\}$, and define

x_i the *fraction* of the file resource stored at node i. Since there is only a single divisible resource, $\Sigma_{i=1}^{n} x_i = 1$. We will assume that accesses are made on a uniform basis (although this can be 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.* The purpose of the distributed resource allocation algorithms is to compute the optimal $(x_1, \cdots, x_n)$.

As discussed in Section I, it is assumed that the resource allocation process is driven using some underlying *model* of the system. We next describe this model and introduce the relevant notation.

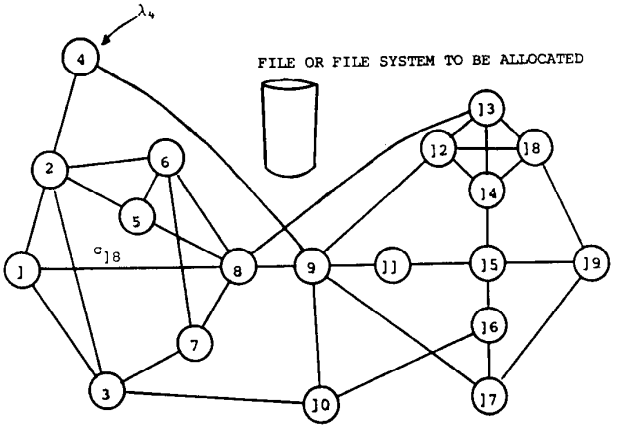


Fig. 1. A distributed computer system.

λ_i the average rate at which node i generates accesses to the file resource. The networkwide access generation rate is defined, $\lambda = \Sigma_{i=1}^{n} \lambda_i$. The arrival rate is modeled as a Poisson process with parameter λ. We note that λ represents the long-term steady-state arrival rate. The arrival rate over different given periods of times may vary (i.e., there will be periods of time in which there are a large number of accesses and other periods of time in which there are much fewer accesses) and it is precisely this *burstiness* that is modeled well by a Poisson process.

c_{ij} the *communication* cost of transmitting an access from node i to node j and transmitting the response from j back to i. c_{ii} is taken to be zero.

C_i the average (systemwide) *communication* cost of making an access at node i. We take this simply as the weighted sum of the individual communication costs:

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

$1/\mu_i$ the average service time for an access request at node i. In order to simplify our presentation, we will also assume $\mu_i = \mu$ for all nodes i and will not distinguish between the service times for queries and updates. As discussed in Section V, each of these restrictions can be easily relaxed. We model the service requirements of each access as an exponential random variable with mean $1/\mu$.

T_i the expected time delay associated with satisfying an access at node i. This delay results both from the queueing and processing time at node i. Given our assumptions concerning λ, x_i, and μ, we have [17]

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

This provides a formula for the long-term average delay at node i, as a function of x_i for a given arrival rate λ and access service rates μ.

The above quantities define the individual communication cost and access time delays. Interestingly, each of these two costs, considered alone, suggest diametrically opposed allocation strategies. If communication is the sole cost, the optimal strategy is simply to put the entire resource at that node i where C_i is minimal. However, concentrating a large amount of a resource at one node means that a correspondingly large number of queries will be directed to that node with resulting large access delays. This would argue for distributing the resource evenly among the n network nodes. In our overall cost function, the relative importance of communication costs and access delays will be characterized by the constant K (in (1) below). The communication cost is normally taken to be the average delay incurred in the transmission of messages and thus, assuming all delays are measured in the same time units, we take $K = 1$ in this paper.

Given the individual costs defined above, the overall expected cost of access to the file resource is given by

$$C = \sum_{i \in N} (\text{cost of access to } x_i)\text{prob(accessing } x_i)$$

$$= \sum_{i \in N} (C_i + KT_i)x_i$$

$$= \sum_{i \in N} \left(C_i + \frac{K}{\mu - \lambda x_i} \right) x_i. \tag{1}$$

Our optimization problem is thus to minimize C subject to the constraints $\Sigma_{i \in N} x_i = 1$, $x_i \geq 0$, $\forall i \in N$, and $\mu > \lambda$. (We note that the weaker assumption $\lambda < \Sigma_{i \in N} \mu_i$ may be made provided that an initial allocation is made such that $\lambda x_i < \mu_i$, for all i.) Equivalently, we can take the negative of the cost function to be our objective function and maximize this function. In order to reflect the microeconomic origins of our optimization algorithms, we adopt this latter problem formulation and refer to the objective function as the *utility* of the file allocation.

$$U = \text{Utility} = -C = -\sum_{i \in N} \left(C_i + \frac{K}{\mu - \lambda x_i} \right) x_i. \tag{2}$$

IV. Decentralized Algorithms for Optimal File Allocation

In this section, we present and examine three decentralized algorithms for optimally allocating the file resource. All three algorithms are variations on the resource-directed approach discussed in Section II; they differ primarily in the amounts and structure of the communication and computation required, and in their convergence speeds.

A. A First Derivative Algorithm

The first algorithm is closely based on Heal's [14] normative model of economic planning and is essentially the algorithm first derived from [14] in [15]. Assume some initial allocation $(x_1, \cdots, x_n)$ of the file resource, where $\Sigma_{i \in N} x_i = 1$. The particular initial allocation selected will not affect the final (optimal) file allocation but will play an important role in determining the convergence speed of the algorithm.

The algorithm is iterative in nature and consists of a local computation followed by a communication step. Each node i 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. This can be done by having all nodes transmit their marginal utility to a central node which computes the average and broadcasts the results back to the individual nodes or each node may broadcast its marginal utility to all other nodes and then each node may compute the average marginal utility locally. (In a broadcast environment, e.g., 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) checks to see if the algorithm termination criteria have been met. If so, the procedure stops and, as discussed below, the resulting allocation is optimal.

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

1) Algorithm Statement: A more precise statement of the algorithm is now given; the notation U_i' is used to denote $\partial U(x_1, \cdots, x_n)/\partial x_i$.

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

DO a) Each node $i \in N$ calculates U_i' evaluated at the current allocation and sends U_i' and x_i to all nodes $j, j \neq i, j \in N$ or to the designated central agent.

b) Each node $i \in A$ (or the central agent) computes its change in allocation by

$$\Delta x_i = \alpha \left(U_i' - \frac{1}{|A|} \sum_{j \in A} U_j' \right), \qquad \forall i \in A \tag{3}$$

where α is the stepsize parameter and A is the set of nodes described below. $\forall i \notin A$, $\Delta x_i = 0$.

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

UNTIL $|U_i' - U_j'| < \epsilon$, $\forall i, j \in A$.

In the above algorithm, we refer to the term which scales the difference between the average marginal utility and an individual marginal utility as the *stepsize parameter*; in this case, the stepsize parameter is simply the constant α. In Theorem 3 in Appendix B, we derive bounds on the size of α to guarantee convergence of the above file reallocation process. The construction of the set A (shown below) is required to ensure that no node's allocation goes below zero as a result of step c) and that nodes receiving a zero allocation

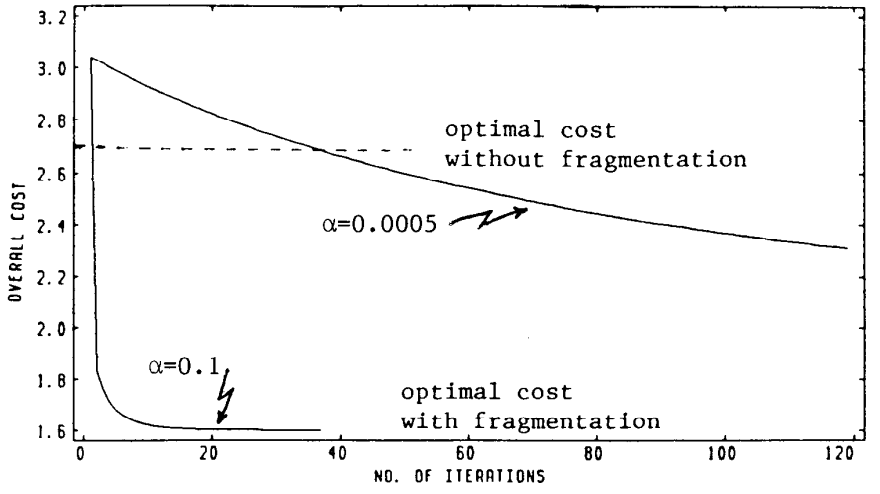


Fig. 2. Access cost versus iteration number for the first derivative algorithm.

have below average marginal utilities. Typically, A is the full set of nodes, N.

/* Algorithm for computing the set A* at each iteration/

i) For all i, sort U_i'.

ii) Set $A' = \{i |$ node i has largest $U_i'\}$.

iii) Do step iv) for each $j, j \notin A'$ in descending order of U_j'.

iv) If j would receive a positive allocation x_j as a result of the reallocation defined by (3) above with $A = A' \cup \{j\}$, then set $A' = A' \cup j$.

v) Set $A = A'$.

2) Algorithm Properties: Optimality, Feasibility, Monotonicity, and Convergence: The above algorithm has several properties that make it particularly attractive for file resource allocation in a distributed environment. The first property is that when the algorithm converges, (i.e., $\partial U/\partial x_i = \partial U/\partial x_j$ $\forall i, j \in A$), the necessary conditions for optimality have been satisfied; a proof of this property can be found in [14]. Although these conditions are also satisfied by local optima, local minima, and points of inflection, the utility function in (2) has no such points 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, we can be assured of a globally optimum allocation.

Another important property of the algorithm (Theorem 1 in Appendix B) is that it maintains a feasible file resource allocation ($\Sigma_{i \in N} x_i = 1$) at each iteration. Thus, the algorithm can be terminated prematurely (before convergence) and the resulting file allocation will be feasible (i.e., the system will be operational, although its performance would not be optimal). The third important property is that as a result of each iteration, $\Delta U > 0$, that is, the systemwide utility function (2) monotonically increases 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 established in Theorem 3 in Appendix B.

The final property is that for sufficiently small values of the stepsize parameter, in this case α, the algorithm can be shown to *converge* to the optimal file allocation. A proof of convergence, initially due to Heal [14] holds for the case of infinitesimally small values of α. In Theorem 3 in Appendix B, we establish an upper bound on the size of a discrete α which guarantees convergence.

3) Experimental Results: Fig. 2 plots the overall cost of the computed file resource allocation as a function of the iteration number for the first derivative algorithm in the network (from [24]) shown in Fig. 1.

In this experiment, the individual link costs were all taken to be 0.024, $K = 1$, $\mu = 1.5$, $\lambda_i = 1.0/n$, $\forall i = 1 \cdots, n$, and the initial allocation was $x_1 = 1.0$ and $x_i = 0.0$, $\forall i \geq 2$. ϵ was chosen such that all the partial derivatives were within one tenth of one percent of each other at convergence (a very stringent convergence criteria). Minimum hop routing was used for determining the cost of routing accesses between nodes. The lower curve in the figure is for executing the algorithm with $\alpha = 0.1$ and the upper curve is for $\alpha = 0.0005$. The optimal cost without fragmentation is also shown.

The figure illustrates several important aspects of the algorithm. First, the figure demonstrates that fragmenting the file resource results in a considerable cost decrease over the integer allocation case. Second, the figure clearly shows the monotonic cost decrease achieved by the algorithm and demonstrates that the algorithm can converge upon the optimal allocation in a small number of iterations.

The figure also demonstrates, however, the importance of selecting an appropriate value of α. For example, with $\alpha = 0.1$ only 36 iterations were required until all partial derivatives were within one tenth of one percent of each other, while with $\alpha = 0.0005$, over 7000 iterations were required. In the

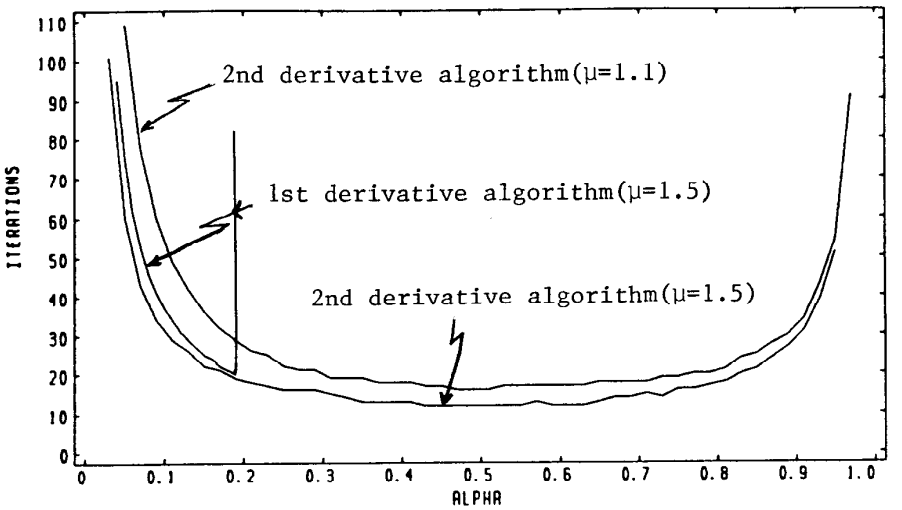


Fig. 3. Convergence times as a function of α.

following subsection, a second derivative algorithm is studied which results in rapid convergence over a significantly wider range of stepsize parameter values.

B. A Second Derivative Algorithm

Intuitively, it should be possible to improve the convergence behavior of the previous algorithm by using a variable stepsize parameter [3] that is sensitive to the current allocation and shape of the utility function rather than the constant value of α in (3). As shown in Appendix A, the gradient algorithm of Section IV-A-1 can be shown to result from optimizing a *first*-order approximation of the change in the utility function (2). An optimization algorithm based on a second-order approximation of (2) permits each node to scale its stepsize parameter by a different amount at each iteration. The additional overhead incurred by this algorithm is the computation and communication of this second derivative information.

1) Algorithm Statement: A more precise statement of the algorithm is now given.

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

DO 1. Each node $i \in N$ calculates U_i' and the inverses of the second partial derivatives, $k_i = 1/|\partial^2 U/\partial x_i^2|$, evaluated at the current allocation and sends U_i', k_i, and x_i to all nodes j, $j \neq i$, $j \in N$ or to the designated central agent.

2. For each node $i \in A$ [where A is the same set as before except that (4) is used in step iv)] the change in its file resource allocation is given by

$$\Delta x_i = \alpha k_i \left(U_i' - \frac{\Sigma_{i \in A} k_i U_i'}{\Sigma_{i \in A} k_i} \right) \tag{4}$$

3. The current allocation for each node $i \in A$ is updated: $x_i := x_i + \Delta x_i$

UNTIL $|U_i' - U_j'| < \epsilon$, $\forall i, j \in A$.

In Appendix B, it is shown that the second derivative algorithm (4) also maintains feasible allocations and produces strictly monotonic increases in utility and again determines a bound on the constant α such that the discrete second derivative algorithm provably converges.

2) Experimental Results: Fig. 3 compares the number of iterations for convergence as a function of the constant α for the first and second derivative algorithms and clearly demonstrates the superiority of the second derivative algorithm. (With the exception of μ, the network parameters are identical to those in Section IV-A-3.) For example, with $\mu = 1.5$, $\lambda = 1.0$, fewer iterations are required by the second derivative algorithm and the algorithm converges over a much wider range of α. Perhaps even more importantly, the bound on α derived in Appendix B is such that it is a relatively simple matter to choose a practical value of α which will result in a guaranteed (and relatively rapid) convergence.

The third curve in Fig. 3 shows algorithmic convergence with $\mu = 1.1$ and $\lambda = 1.0$. Note that for these parameter values, the utility function (2) is quite steep and yet the algorithm still demonstrates rapid convergence. The corresponding behavior of the first derivative algorithm (not shown) was considerably worse, at *best* requiring 241 iterations and again being very sensitive to the value of α chosen.

Finally, Fig. 4, compares the convergence behavior of the above algorithm (algorithm $A1$ in Fig. 4) to the performance of an alternate second derivative algorithm ($A2$ in Fig. 4) derived from an approach used in [3] and [13] for optimizing routing assignments in computer communication networks. Note that the algorithm of (4) is the more stable of the two with respect to both changes in α as well as the shape of the utility function (as determined, in this case, by the service rate μ). We conjecture that this results from the fact that for larger values of α (where faster convergence is achieved), the reallocation mechanism derived from [3] and [13] makes a positive reallocation to just one node at each iteration (that node with the highest marginal utility) and the node receiving this positive reallocation will change from one iteration to the

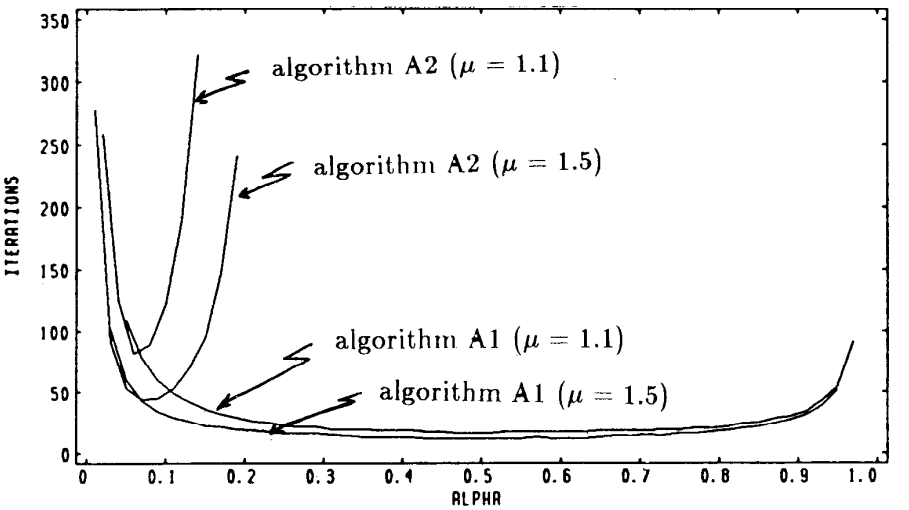


Fig. 4. Comparision of convergence behavior of two second derivative algorithms.

next. The algorithm under study in this paper, however, makes a more balanced reallocation at each iteration.

C. A Pairwise Interaction Algorithm

During each iteration of the previous two algorithms, all nodes perform a simple gradient calculation (in parallel) using local information followed by a *networkwide* exchange of information. If all partial derivatives are reported to a central agent, $O(n)$ messages are required at each iteration; if partial derivatives are broadcast to all nodes, $O(n^2)$ messages are required in a nonbroadcast environment. Since in many cases, the communication overhead may dominate the time required by each iteration, it is of interest to investigate algorithms which decrease this communication overhead while incurring only a slight increase in the computation (number of iterations) required at each node.

The third algorithm studied is based on *localized* exchange of gradient information among neighbors, a problem also considered earlier in [15] in which a symmetric communication structure among nodes was assumed. In this algorithm, the iterative process is refined into *major* and *minor* iterations. At each minor iteration, the nodes in the network are paired according to a pairing relation (see below). Each pair of nodes then computes a pairwise reallocation based on their difference in marginal utilities. Several minor iterations, each governed by a pairing relation, are performed sequentially and constitute a major iteration. We note that numerous pairs may proceed concurrently with their pairwise exchanges and that considerable asynchrony may thus be achieved. We now define the following terms:

$\mathcal{P}$—an *unordered pairing relation*, $\mathcal{P} \subset N \times N$, such that if $(i_1, i_2), (i_3, i_4) \in \mathcal{P}$ then $i_1 \neq i_2 \neq i_3 \neq i_4$. Stated more simply, each node must belong to at most one pair.

ℓ—the number of minor iterations in a major iteration.

$\mathfrak{M}$—a *major* iteration—a sequence of ℓ pairing relations, $\mathfrak{M} = \langle \mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_\ell \rangle$, each of which defines a *minor* iteration, and such that for any pair of two distinct nodes $(i_1, i_2) \in N \times N$, $(i_1, i_2) \in (\cup_{1 \leq j \leq \ell} \mathcal{P}_j)^+$, the transitive closure of the union of all the pairing relations. This restriction ensures that portions of the file can eventually be transferred between any two nodes in the network given a sufficient number of iterations.

A *neighbor* of a node i is a node j such that $(i, j) \in \mathcal{P}$ for at least one $\mathcal{P} \in \mathfrak{M}$.

The pairwise algorithm can be derived mathematically following the same arguments as those presented in Appendix A for the previous two algorithms. We construct the sequence of minor iterations by solving ℓ problems of the form

$$\max \Delta U$$

$$\text{subject to } \Delta x_i + \Delta x_j = 0, \qquad \forall (i, j) \in \mathcal{P}_k$$

for each pairing relation $\mathcal{P}_k \in \mathfrak{M}$, $k = 1, \cdots, \ell$. The resulting pairwise reallocation process at the kth minor iteration can be shown to be

$$\Delta x_i = \frac{\alpha k_i k_j}{k_i + k_j} (U_i' - U_j') \qquad \forall (i, j) \in \mathcal{P}_k \tag{5}$$

where α is again a constant and $k_i = 1/|\partial^2 U/\partial x_i^2|$. Thus, at each minor iteration every pair of nodes in the pairing relation performs an exchange of file resource according to (5). Note that the algorithm requires communication only between direct neighbors.

A sequence of minor iterations corresponding to a major iteration is executed and the process is repeated until convergence. The termination criteria are identical to those in our previous algorithms but are slightly more complicated to implement in practice. Periodically, all nodes must be polled to determine whether their marginal utility equals that of each of their neighbors (assuming both have a nonzero allocation). If this condition is true for all nodes, the algorithm terminates since by the transitive closure property, it must be true that the nodes have equal marginal utilities, the necessary condition for an optimum.

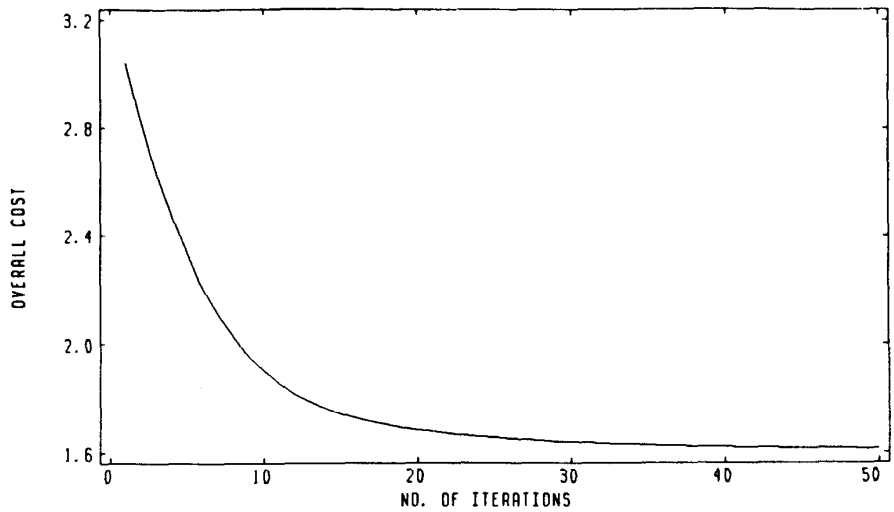


Fig. 5. Access cost versus iteration number for the pairwise algorithm.

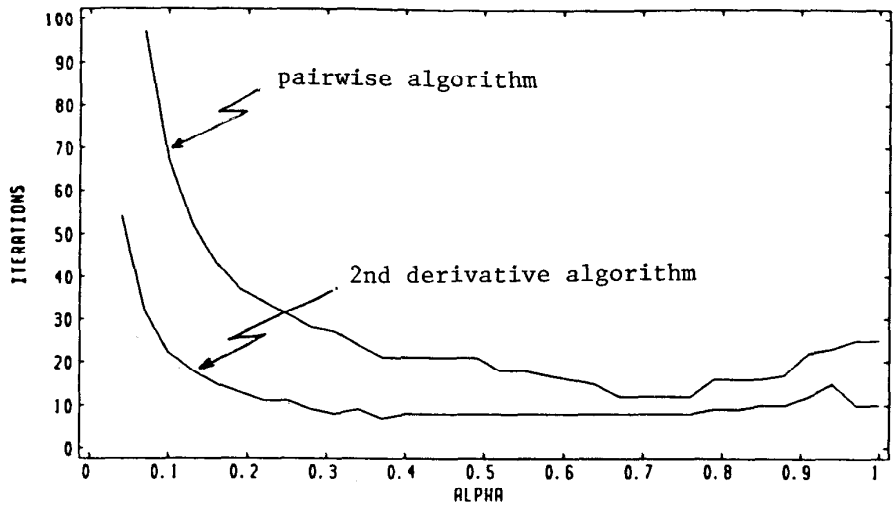


Fig. 6. Convergence time comparison of pairwise and second derivative algorithms.

1) Experimental Results: The pairwise algorithm was studied using the same network parameters as in Sections IV-A-3 and IV-B-2 and the three pairing relations—$\mathcal{P}_1 = \{(1, 2), (5, 6), (7, 8), (3, 10), (4, 9), (15, 16), (17, 19), (12, 13), (14, 18)\}$, $\mathcal{P}_2 = \{(2, 4), (1, 3), (6, 7), (8, 13), (12, 14), (11, 15), (9, 17), (10, 16)\}$, $\mathcal{P}_3 = \{(1, 8), (2, 5), (3, 7), (9, 12), (14, 15), (13, 18), (17, 19), (10, 16)\}$.

Fig. 5 plots the file resource access cost as a function of the number of minor iterations and shows the same convergence behavior as the previous two algorithms: a rapid performance increase to a region around the optimum in relatively few iterations and then a slower approach towards the optimum. We observed, however, that when ϵ is very small, the pairwise algorithm caused small portions of the file to thrash from one node to another during the final approach to the optimum, resulting in longer convergence times. For these studies, ϵ was thus chosen so that the convergence criterion was that the partial derivatives be within five percent of each other. In this case, the allocation upon which the algorithm converged was insignificantly different from the true optimal allocation.

Fig. 6 plots the total number of (minor) iterations required for convergence as a function of α for the second derivative algorithm and the pairwise algorithm. Note that although the pairwise algorithm requires up to twice as many iterations to converge, each iteration requires only a *constant* order of communication (due to the pairwise communication structure of the algorithm) rather than the $O(n)$ or $O(n^2)$ communication required by the second derivative algorithm. This would indicate that a pairwise algorithm would perform well in all but the smallest of distributed systems.

Finally, with pairwise interaction, it now becomes practicable to have each set of paired nodes *search* for a more effective value of α during an iteration. For example, let α_c be some initial (fixed) small value for α and let the subscript m be defined such that $U'_m = \min(U'_i, U'_j)$ for two paired nodes i and j. If $|\Delta x_i| = |\Delta x_j|$ is the amount of resource exchanged

between nodes i and j at the current value of α, then define the endpoints of a line search to be

$$\text{lower limit} = \min\left(\alpha_c, \frac{x_m}{|\Delta x_i|}\right)$$

$$\text{upper limit} = \max\left(\alpha_c, \frac{x_m}{|\Delta x_i|}\right)$$

During an interaction, each interacting pair of nodes may then conduct a binary search of a predetermined number of iterations within this interval to attempt to locate a value of α that results in a one-step reallocation (5) which maximizes the sum of the utilities of nodes i and j. Our simulation studies (not reported here) have shown that using even two iterations of this process may significantly accelerate convergence.

V. Discussion

Having examined three decentralized algorithms for allocating a file resource, we now discuss several issues relating to these algorithms.

Generalizations: Let us first consider extending our previous formulation of FAP to more general file resource allocation problems. 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 (2) and weighting these costs appropriately. Different access processing rates can also be trivially incorporated by replacing the μ in (2) by the individual μ_i's. 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, although the convergence criteria on the size of α would have to be established for these new objective functions.

In the case of a single copy of M multiple distinct file resources, the utility function can be easily extended by introducing variables for additional resources, x^j, $j \in \{1 \cdots M\}$. We may now define x_i^j as the fraction of file resource j allocated to node i, define λ^j as the access rate to this resource, and redefine the objective function using these variables in a straightforward manner. Unfortunately, the extensions for multiple copies of a file resource are not as simple. In the single copy case, knowing the *fraction* of the resource allocated to a node was sufficient to determine the rate at which it would receive access requests from the other nodes in the system. In the most general case of multiple copies, the *contents* (records in the case of file allocation, or files in the case of file system allocation) of the fraction of file resources allocated to a node must be considered in determining these rates.

One possible solution to this problem is to impose additional structure onto the system. In particular, we may order the network nodes into a unidirectional *virtual ring* with copies of the file laid out contiguously around the ring. If each node is constrained to direct its accesses in a given direction along this virtual ring, a node may easily determine the rate at which it will receive accesses. Our experiments have indicated, however, that this problem formulation may result in nonmonotonic changes in the cost function at successive iterations, particularly as the algorithm nears convergence. One way to mitigate these effects is to decrease the value of α, although as α is decreased, the number of iterations needed to converge to within a region around the optimum will increase.

Roundoff Analysis: In the previous discussion, it was assumed that a file resource is an infinitely divisible resource. In practice, however, the file resource may only be broken on atomic boundaries (a record in the case of a file or a file in the case of a file system). Thus, the real-number fractions determined as a result of the optimization process will have to be rounded so that the file resource will fragment at atomic boundaries.

To provide a worst case analysis of the error introduced by this process, let x_i' be the rounded fraction for node i. Then $|x_i - x_i'|$ is the magnitude of the roundoff error for each i, and the error in the overall resulting utility can be approximated to first order by $\Delta U_{\text{err}} \approx \Sigma_{i \in N} U_i' |x_i - x_i'|$. When each fraction is rounded to the nearest atomic unit boundary, the error in that particular fraction will be less than one atomic unit. If there are R atomic units in a resource, the maximum value of $|x_i - x_i'|$ is $1/R$, the worst case error in utility is given by

$$\Delta U_{\text{err}} \leq \frac{1}{R} \sum_{i \in N} U_i'$$

and the worst case error tends to zero as the number of records tends to infinity. Our experiments with the network of Fig. 1 resulted in a five percent roundoff with a 20 atomic-unit resource, decreasing to a 1.5 percent error with 60 units and to less than 1 percent error for a resource with a hundred atomic units.

Integration with Higher Level Mechanisms: In a distributed system, the layer of software that handles any resource typically serves higher level layers that, for consistency, may require operations on data to have properties of atomicity and serializability [12]. When fragmentation of a resource is permitted, the effects of such fragmentation on these higher level mechanisms should be considered.

If locking is required on the basis of atomic units, then fragmentation incurs no additional overhead for supporting the higher level mechanisms. However, if an entire file resource needs to be locked and is fragmented over several nodes, additional message overhead is required to ensure the serializability of concurrent transactions, to avoid deadlock, and/or preserve atomicity. Thus, there do exist situations in which file fragmentation is not desirable. However, if these operations are relatively infrequent, fragmentation provides an attractive alternative to integral file allocation policies from both a performance and reliability point of view.

VI. Conclusions and Future Work

In this paper, we have presented several algorithms which provide simple, decentralized procedures for distributing a resource in a distributed system. The problem of file resource allocation was studied in order to provide a context in which these algorithms could be considered. These algorithms were developed using models of resource allocation previously

developed within the field of mathematical economics and were shown to have the attractive properties of maintaining feasibility, strict monotonicity, and fast convergence. These properties were both formally established and studied through simulation.

The importance of monotonicity arises in distributed operating sytems 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 whenever the system is lightly loaded in order to gradually improve the allocation.

Our algorithm was based on the use of an underlying *model* to predict system performance for a given resource allocation. The possibility also exists of using observed *measurements* of gradient information [5] to control the reallocation process. We note, however, that there will be uncertainty associated with these measurements and thus stochastic approximation techniques will be required in the optimization process. We believe this is an important problem and a promising direction for future research.

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 many. The methods are well understood and tested already. 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 may reduce the complexity of designing and implementing the large distributed systems of the future.

Appendix A

The Framework for Gradient Algorithms

The approach towards designing our algorithms is based on the gradient projection method [19] with the addition of constructing the feasible set A. This technique has been previously employed in the area of computer networks [3] and general resource allocation [15]. An iterative resource allocation algorithm is defined as a list of expressions Δx_i, one for each node in the distributed system (i.e., in the same manner as (3) defines an algorithm). We obtain a particular algorithm by expressing the change in utility ΔU (resulting from a reallocation of file resources) as a function of the individual Δx_i's and then solving the problem

$$\text{maximize: } \Delta U = \sum_{i \in N} U_i' \Delta x_i + \frac{1}{2!} \sum_{i \in N} U_i'' \Delta x_i^2 + \frac{1}{3!} \sum_{i \in N} U_i''' \Delta x_i^3 + \cdots \quad (6)$$

subject to the given feasibility constraints. In solving the above maximization problem, we obtain expressions for Δx_i and consequently, an algorithm. Note that there are no cross partial derivatives $\partial^2 U / \partial x_i x_j$ in the Taylor series expansion of ΔU. This is true of the file resource allocation cost model described earlier but need not be the case with other utility functions.

When only limited derivative information is available, an iterative process is generated by solving the above maximization problem. For example, the second derivative algorithm studied in this paper is obtained using a second-order Taylor expansion of ΔU and solving

$$\text{maximize } \sum_{i \in N} U_i' \Delta x_i + \frac{1}{2} \sum_{i \in N} U_i'' \Delta x_i^2 \quad (7)$$

subject to the constraint $\Sigma_{i \in N} \Delta x_i = 0$. Note that the inequality constraints $x_i \geq 0$ are enforced by the computation of the set A. Using the Lagrange multiplier technique to solve for each individual Δx_i, we obtain the second derivative algorithm of (4), where the stepsize parameter α is used in discretizing what would otherwise be a continuous process. Note that the first derivative algorithm may be obtained by taking the second-order Taylor expansion in (7) and setting k_i equal to unity.

Appendix B

Feasibility and Monotonicity

The theorems below demonstrate that the algorithms studied in this paper maintain a feasible file resource allocation at each iteration while improving, monotonically, the value of the utility. Upper bounds are also obtained for a value of α which guarantees convergence. These theorems are proved for the second derivative algorithm; at the end of each proof we describe briefly how the proofs may be modified for the first derivative and pairwise algorithms. Theorems 1 and 2 below were proven for the first derivative algorithm in [14]; we follow this approach in establishing the theorems for the second derivative and pairwise exchange algorithms.

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

Proof: To prove that a feasible allocation is maintained we show that the sum total change in allocation remains zero [14]. From (4),

$$\begin{aligned} \sum_{i \in N} \Delta x_i &= \sum_{i \in A} \alpha k_i \left(U_i' - \frac{\Sigma_{j \in A} k_j U_j'}{\Sigma_{j \in A} k_j} \right) \\ &= \sum_{i \in A} \alpha k_i U_i' - \sum_{i \in A} \alpha k_i \left(\frac{\Sigma_{j \in A} k_j U_j'}{\Sigma_{j \in A} k_j} \right) \\ &= \sum_{i \in A} \alpha k_i U_i' - \sum_{j \in A} \alpha k_j U_j' \frac{\Sigma_{i \in A} k_i}{\Sigma_{j \in A} k_j} \\ &= 0. \end{aligned}$$

Setting each $k_i = 1$ provides a proof of feasibility for the first derivative algorithm. For the pairwise algorithm, for each

$(i, j) \in \mathcal{P}$ we have $\Delta x_i + \Delta x_j = 0$ and hence feasibility is maintained.

Theorem 2: When the changes in allocation are infinitesimally small, the second derivative algorithm results in a strictly monotonic increase in utility except when the necessary conditions for optimality are satisfied.

Proof: Following [14], we show that the rate of change for a first-order approximation of an *arbitrary* utility function is greater than zero, except when the necessary conditions for optimality are satisfied.

$$\begin{aligned}
\dot{U} &= \sum_{i\in A} U_i' k_i \left(U_i' - \frac{\Sigma_{j\in A} k_j U_j'}{\Sigma_{j\in A} k_j} \right) \\
&= \sum_{i\in A} k_i (U_i')^2 - \frac{(\Sigma_{i\in A} k_i U_i')^2}{\Sigma_{i\in A} k_i} \\
&= \sum_{i\in A} k_i (U_i')^2 - 2\frac{(\Sigma_{i\in A} k_i U_i')^2}{\Sigma_{i\in A} k_i} + \frac{(\Sigma_{i\in A} k_i U_i')^2}{\Sigma_{i\in A} k_i} \\
&= \sum_{i\in A} k_i (U_i')^2 - 2\left(\frac{\Sigma_{j\in A} k_j U_j'}{\Sigma_{j\in A} k_j} \right)\left(\sum_{i\in A} k_i U_i' \right) \\
&\quad + \left(\sum_{i\in A} k_i \right)\left(\frac{\Sigma_{j\in A} k_j U_j'}{\Sigma_{j\in A} k_j} \right)^2 \\
&= \sum_{i\in A} k_i \left((U_i')^2 - 2U_i' \frac{\Sigma_{j\in A} k_j U_j'}{\Sigma_{j\in A} k_j} \right. \\
&\quad \left. + \left(\frac{\Sigma_{j\in A} k_j U_j'}{\Sigma_{j\in A} k_j} \right)^2 \right) \\
&= \sum_{i\in A} k_i \left(U_i' - \frac{\Sigma_{j\in A} k_j U_j'}{\Sigma_{j\in A} k_j} \right)^2
\end{aligned}$$

which is greater than zero, with equality when all the U_i' are equal. Note that we require that $\forall i$, $k_i \neq 0$ for a second derivative algorithm to be used. Setting each $k_i \neq 1$ results in a proof of monotonicity for the first derivative algorithm [14]. In the case of the pairwise algorithm, we note that each pair of nodes behaves like a two-node network executing the second derivative algorithm.

Also note that in the proof of Theorem 2, the following identity, which will be used shortly, was also established:

$$\sum_{i\in A} U_i' k_i \left(U_i' - \frac{\Sigma_{j\in A} k_j U_j'}{\Sigma_{j\in A} k_j} \right) = \sum_{i\in A} k_i \left(U_i' - \frac{\Sigma_{j\subset A} k_j U_j'}{\Sigma_{j\in A} k_j} \right)^2. \quad (8)$$

Although the above theorem holds for arbitrary utility functions, it essentially requires that α be infinitesimally small. In the theorem below, we derive a more realistic upper bound on α for the particular problem of file resource allocation.

Theorem 3: for the cost function of (1), there exists a value of α, depending on the parameters of the problem, for which each iteration results in strict monotonic increase in utility, except when the algorithm termination conditions hold.

Proof (first and second derivative algorithms): Our goal is to determine conditions on the value of α under which the overall change in systemwide utility ΔU will be greater than zero as a result of the reallocations $\{\Delta x_i\}$ made at each iteration. We begin by expressing the overall change in utility as the result of each iteration as a Taylor series:

$$\Delta U = \Sigma_{i\in A} U_i' \Delta x_i + \frac{1}{2!} \Sigma_{i\in A} U_i'' (\Delta x_i)^2 + \frac{1}{3!} \Sigma_{i\in A} U_i''' (\Delta x_i)^3 + \cdots. \quad (9)$$

Noting that

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

(9) may be rewritten:

$$\Delta U = \sum_{i\in A} U_i' \Delta x_i + \frac{1}{2} \sum_{i\in A} U_i'' \Delta x_i^2 \left(1 + \left(\frac{\lambda \Delta x_i}{\mu - \lambda x_i} \right) + \left(\frac{\lambda \Delta x_i}{\mu - \lambda x_i} \right)^2 + \left(\frac{\lambda \Delta x_i}{\mu - \lambda x_i} \right)^3 + \cdots \right). \quad (10)$$

In order for the series within the second sum to converge, we must have

$$\left| \frac{\lambda \Delta x_i}{\mu - \lambda x_i} \right| < 1 \quad (11)$$

for all i. A sufficient condition to satisfy this inequality is

$$1 + |\Delta x_i| \leq \frac{\mu}{\lambda} \quad (12)$$

for each i. Recall from (4) that for the second derivative algorithm

$$\Delta x_i = \alpha k_i \left(U_i' - \frac{\Sigma_{i\in A} k_i U_i'}{\Sigma_{i\in A} k_i} \right) \quad (13)$$

and that taking $k_i = 1$ yields the first derivative algorithm. Substituting (13) into inequality (12) and taking bounds on its components gives the following upper bound on the value of α:

$$\alpha < \frac{\mu - \lambda}{\lambda q} \quad (14)$$

where

$$q = \max_i \left(k_i \left(U_i' - \frac{\Sigma_{i\in A} k_i U_i'}{\Sigma_{i\in A} k_i} \right) \right).$$

And thus for the first derivative algorithm

$$q = \left| (C_{\max} - C_{\min}) + \frac{\lambda K (2\mu - \lambda)}{\mu(\mu - \lambda)^2} \right|$$

and for the second derivative algorithm

$$q=\left(\frac{\mu^2}{2\lambda K}\right)\left|(C_{\max}-C_{\min})+\frac{\lambda K(2\mu-\lambda)}{\mu(\mu-\lambda)^2}\right|.$$

The above bound on α guarantees that the Taylor series expansion of ΔU converges. However, we must still ensure that $\Delta U > 0$. By (8), the first term of the series in (10) is positive and thus, a *sufficient* condition to ensure that $\Delta U > 0$ is

$$\sum_{i\in A} U_i'\Delta x_i > \frac{1}{2}\sum_{i\in A}\left|U_i''\Delta x_i^2\frac{1}{1-\left(\frac{\lambda\Delta x_i}{\mu-\lambda x_i}\right)}\right|.$$

which itself will hold when

$$\sum_{i\in A} U_i'\Delta x_i > \frac{1}{2\left(1-\frac{\alpha\lambda}{\mu-\lambda}q\right)}\sum_{i\in A}|U_i''|\Delta x_i^2.$$

Substituting for Δx from (13) and using (8) to replace the sum containing U_i', we get after some manipulation

$$\alpha < \frac{2}{\frac{2\lambda}{\mu-\lambda}q+\max_i|U_i''k_i|}. \tag{15}$$

It may be easily verified that the upper bound on α in inequality (15) is always stricter than that of inequality (14) and thus inequality (15) provides an upper bound on α that ensures that $\Delta U > 0$.

Finally, Theorem 4 establishes the fact that each increase in utility is always bounded below by some finite value. Together with Theorem 3 (which states that the increase in utility is always nonnegative) this ensures that the algorithm will not produce an infinite sequence of allocations which results in a sequence of changes in utility which become increasingly smaller and smaller. Since the utility of an optimal allocation is finite, this then guarantees that the file resource allocation algorithm eventually converges.

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

Proof: Providing that we choose the value of α to satisfy inequality (15) we have from (10) that

$$\Delta U=\sum_{i\in A} U_i'\Delta x_i+\frac{1}{2}\sum_{i\in A}U_i''\Delta x_i^2\frac{1}{1-\left(\frac{\lambda\Delta x_i}{\mu-\lambda x_i}\right)}$$

is nonnegative. Substituting for Δx_i from (13) and rearranging terms using Theorem 2, we get

$$\Delta U=\sum_{i\in A}(\alpha k_i)\left(1-\frac{\alpha|U_i''k_i|}{2\left(1-\frac{\lambda\Delta x_i}{\mu-\lambda x_i}\right)}\right)\cdot\left(U_i'-\frac{\Sigma_{j\in A}k_jU_j'}{\Sigma_{j\in A}k_j}\right)^2. \tag{16}$$

Rearranging inequality (15) we have

$$\frac{\alpha\max_i|U_i''k_i|}{2\left(1-\frac{\lambda}{\mu-\lambda}\alpha q\right)}<1 \tag{17}$$

which, by (11), must also be greater than zero. We thus have

$$\Delta U\geq(\alpha\min_i(k_i))\left(1-\frac{\alpha\max_i|U_i''k_i|}{2\left(1-\frac{\lambda}{\mu-\lambda}\alpha q\right)}\right)\cdot\sum_{i\in A}\left(U_i'-\frac{\Sigma_{j\in A}k_jU_j'}{\Sigma_{j\in A}k_j}\right)^2. \tag{18}$$

Finally, it may be easily verified that given that the algorithm has not converged, the sum in (18) is minimized when all U_l', $l\in A$ are equal to $\Sigma_{i\in A}k_iU_i/\Sigma_{i\in A}k_i$, $l\neq i\neq j$ and U_i' and U_j' each lie a distance $\epsilon/2$ from this weighted average. We thus have after evaluating the various terms

$$\Delta U\geq\alpha\frac{(\mu-\lambda)^3}{2K\mu\lambda}\left(1-\frac{\alpha\max\left(1,\frac{2\mu\lambda K}{(\mu-\lambda)^3}\right)}{2\left(1-\frac{\lambda}{\mu-\lambda}\alpha q\right)}\right)\frac{\epsilon^2}{2}>0. \tag{19}$$

References

[1] C. Agnew, "The dynamic control of congestion prone systems through pricing," Ph.D. dissertation, Stanford Univ., Nov. 1973.

[2] K. Arrow and F. Hahn, *General Competitive Analysis*. San Francisco, CA: Holden Day, 1971.

[3] D. Bertsekas, E. Gafni, and R. Gallager, "Second derivative algorithms for minimum delay distributed routing in networks," *IEEE Trans. Commun.*, vol. COM-32, pp. 911-919, Aug. 1984.

[4] D. Bertsekas and R. Gallager, *Data Networks*. Englewood Cliffs, NJ: Prentice-Hall, 1987.

[5] C. G. Cassandras and Y. C. Ho, "An event domain formalism for sample path perturbation analysis of discrete event dynamic systems," *IEEE Trans. Automat. Contr.*, vol. AC-30, no. 12, 1985.

[6] R. Casey, "Allocation of copies of files in an information network," *AFIPS Proc.*, vol. 40, pp. 617-625, 1973.

[7] S. Ceri, G. Pelagatti, and G. Martella, "Optimal file allocation in a computer network: A solution based on the knapsack problem," *Comput. Networks*, vol. 6, pp. 345-357, 1982.

[8] P. Chen, "Optimal file allocation in multilevel storage systems," *Proc. AFIPS*, vol. 42, no. 2, pp. 277-282, 1973.

[9] W. W. Chu, "Optimal file allocation in a multiple computer system," *IEEE Trans. Comput.*, vol. C-18, pp. 885-889, 1969.

[10] L. Dowdy and D. Foster, "Comparative models of the file assignment problem," *ACM Comput. Surveys*, vol. 14, pp. 287 314, June 1982.

[11] K. Eswaran, "Placement of records of a file and file allocation in a computer network," *IFIP Conf. Proc.*, pp. 304-307, 1974.

[12] K. Eswaran, J. Gray, R. Lorie, and I. Traiger, "The notions of consistency and predicate locks in a database system," *Commun. ACM*, pp. 624-633, Nov. 1976.

[13] R. G. Gallager, "A minimum delay routing algorithm using distributed computation," *IEEE Trans. Commun.*, pp. 71-85, Jan. 1977.

[14] G. Heal, "Planning without prices," *Rev. Econ. Studies*, vol. 36, pp. 346-362, 1969.

[15] Y. C. Ho, L. Servi, and R. Suri, "A class of center-free resource allocation algorithms," *Large Scale Syst.*, vol. 1, pp. 51-62, 1980.

[16] L. Hurwicz, "The design of mechanisms for resource allocation," *Amer. Econ. Rev.*, vol. 63, no. 2, pp. 1-30, 1973.

[17] L. Kleinrock, *Queueing Systems: Vol. 1: Theory*. New York: Wiley-Interscience, 1975.

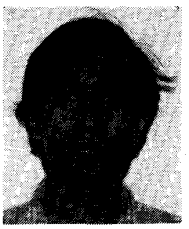
[18] J. F. Kurose, M. Schwartz, and Y. Yemini, "A microeconomic approach to optimization of channel access policies in multiaccess

networks," in *Proc. 5th Int. Conf. Distrib. Comput. Syst.*, Denver, CO, May 1985, pp. 70–80.
[19] E. S. Levitin and B. T. Polyak, "Constrained minimization problems," *USSR Comput. Math. Math. Phys.*, vol. 6, pp. 1–50, 1966.
[20] M. Mahmoud and J. Riordan, "Optimal allocation of resources in distributed information networks," *ACM Trans. Database Syst.*, vol. 1, no. 1, pp. 66–78, 1976.
[21] C. V. Ramamoorthy and K. M. Chandy, "Optimization of memory hierarchies in miltiprogrammed systems," *J. ACM*, vol. 17, pp. 426–445, July 1970.
[22] J. Rothnie and N. Goodman, "A survey of research and development in distributed database management," in *Proc. 1977 Conf. Very Large Databases*, 1977.
[23] R. Suri, "A decentralized approach to optimal file allocation in computer networks," in *Proc. 18th IEEE Conf. Decision Contr.*, IEEE Press, 1979, pp. 141–146.
[24] G. H. Thaker and J. B. Cain, "Interactions between routing and flow control algorithms," *IEEE Trans. Commun.*, vol. COM-34, Mar. 1986.
[25] B. Wah, "File placement in distributed computer systems," *IEEE Computer*, vol. 17, pp. 23–33, Jan. 1984.
[26] M. Wasan, *Stochastic Approximation, Cambridge Tracts in Mathematics and Mathematical Physics No. 58*. Cambridge, MA: Cambridge, Univ. Press, 1969.
[27] M. Woodside and S. Tripathi, "Optimal allocation of file servers in a local area network," *IEEE Trans. Software Eng.*, vol. SE-12, pp. 844–845, 1986.

James F. Kurose (S'81–M'84) received the B.A. degree in physics from Wesleyan University, Middletown, in 1978 and the M.S. and Ph.D. degrees in computer science from Columbia University, New York, NY, in 1980 and 1984, respectively.

Since 1984, he has been an Assistant Professor in the Department of Computer and Information Science, University of Massachusetts, Amherst, where he currently leads several research efforts in the areas of computer communication networks, distributed systems, and modeling and performance evaluation.

Dr. Kurose is a member of Phi Beta Kappa, Sigma Xi, and the Association for Computing Machinery.



Rahul Simha received the B.S. degree in computer science from the Birla Institute of Technology and Science (BITS), Pilani, India, in 1984 and the M.S. degree in computer science from the University of Massachusetts, Amherst, in 1985.

He is currently working towards the Ph.D. degree and his research interests include distributed systems, computer networks, and stochastic optimization.