OPTIMAL ALLOCATION OF MULTIPLE CLASS RESOURCES IN COMPUTER SYSTEMS

A.N. Tantawi, D. Towsley and J. Wolf

COINS Technical Report 88-09

February 1, 1988

To appear in Proceedings of Sigmetrics Conference 1988.

OPTIMAL ALLOCATION OF MULTIPLE CLASS RESOURCES IN COMPUTER SYSTEMS [1]

A.N. Tantawi[2], D. Towsley[3], and J. Wolf[2]

Abstract

A class-constrained resource allocation problem is considered. In this problem, a set of M heterogeneous resources is to be allocated optimally among a set of L users belonging to K user classes. A set of class allocation constraints, which limit the number of users of a given class that could be allocated to a given resource, is imposed. An algorithm with worst case time complexity $O(M(LM + M^2 + LK))$ is presented along with a proof of its correctness. This problem arises in many areas of resource management in computer systems, such as load balancing in distributed systems, transaction processing in distributed database systems, and session allocation in time-shared computer systems. We illustrate the behavior of this algorithm with an example where file servers are to be allocated to workstations of multiple classes.

1 Introduction

A well-known resource allocation problem is the problem of allocating M heterogeneous resources to a set of L users so that the total throughput is maximised. Formally, the problem is stated as: maximise $\sum_{m=1}^{M} \mu_m(x_m)$ such that $\sum_{m=1}^{M} x_m = L$, where x_m, $m = 1, 2, \cdots, M$, is a non-negative integer denoting the number of users allocated to resource m and $\mu_m(x_m)$ is a concave function denoting the throughput of resource m given x_m users allocated to it. This problem is related to the selection problem [7] where the L^{th} largest element in a set S, of size s, given in the form of M pairwise disjoint subsets, is to be found. Several efficient algorithms exist for the selection problem, and alternatively for the resource allocation problem. The earliest class of algorithms is incremental in nature, such as Fox's algorithm [6] which has a time complexity of $O(M + L \log M)$. In [7], a more efficient algorithm with complexity $O(M \log^2(s/M))$ is presented. For the resource allocation problem, the size s of the set S corresponds to the product LM, and thus this algorithm is $O(M \log^2 L)$. An algorithm which finds the K best solutions to the resource allocation problem is provided in [8]; it has a time complexity of $O(C^* + K \log K + K\sqrt{M \log M})$, where C^* is the time complexity to obtain the best solution. A very efficient algorithm requiring only $O(M \log L)$ time is given in [5].

An application of this problem results from the allocation of file servers to workstations in a local network environment [12,13]. In this case, resources and users correspond to file servers and workstations, respectively. It is assumed that all the files accessed by a workstation are stored on the same file server and that the communication delays are negligible. Under these assumptions, and for a set of heterogeneous file servers, one is interested in the number of workstations to be allocated to each file server so that the overall system throughput is maximised.

Another application belongs to the area of static load balancing in heterogeneous multi-computer systems. Here, resources and users correspond to host computers and sessions (or active users), respectively. Assuming the existence of a fast switching mechanism between users and host computers, one is interested in finding the optimal number of sessions to be allocated to a host computer so as to maximise system throughput. This gives rise to an integer programming problem similar to the resource allocation problem. A continuous version of this problem, where the load on a host computer is modeled by an arrival stream of jobs,

[1] This work was supported in part by the National Science Foundation under grant ECS-8406402.

[2] IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598.

[3] Dept. Computer and Information Sciences, U. Massachusetts, Amherst, MA 01003.

is considered in [10,11] and a multiple class version of the problem in [1].

In many applications, one is typically faced with the fact that users belong to different classes and that the number of users from a certain class to be allocated to a resource is limited, sometimes to zero. For example, data requests generated from a workstation may be served only by the subset of file servers which have copies of the data requested. File servers which do not have a copy of that data cannot be allocated to that particular workstation. This gives rise to an extension of the original resource allocation problem to include classes of users and a set of constraints on the number of users from a given class that could be allocated to a given resource. We refer to this problem as the class-constrained resource allocation problem. A closely related problem is considered in [3] where two algorithms are presented: a marginal allocation algorithm and an augmenting path algorithm. Both algorithms are based on computing maximal flows and are applied to a bidding model for oil and gas ventures [4] . Our approach to solving the class-constrained resource allocation problem consists of solving recursively a set of smaller problems which we introduce.

The paper is organised as follows. A formulation of the class-constrained resource allocation problem is presented in Section 2. The algorithm is given in Section 3 and its correctness proved in Section 4. In Section 5, we obtain the time complexity of the algorithm. Computer systems applications are provided and discussed in Section 6.

2 Problem Formulation

Consider a finite set of M servers $\mathcal{M}$ that serves a finite population of L users. These users are divided into a set of classes $\mathcal{K} = \{1, 2, \cdots, K\}$ with L_k users in the k^{th} class, $k \in \mathcal{K}$. The throughput of server m is a function of the number of users x_m that are assigned to it and is denoted by $\mu_m(x_m)$, $x_m = 0, 1, \cdots$; $m \in \mathcal{M}$. We shall assume that $\mu_m(x_m)$ is a concave function of x_m and that it does not depend on which classes of users it is serving. Last, we impose constraints on how users can be assigned to servers. Let $N_m \geq 0$ be the maximum number of users that can be assigned to server m, $m \in \mathcal{M}$. Let $N_{m,k} \geq 0$ be the maximum number of class k users that can be assigned to server m. These constraints may exist because of memory limitations associated with each server. They may also arise because a class k user requires a particular kind of resource that is present at server m, $(N_{m,k} = L_k)$ but is absent at server m', $(N_{m',k} = 0)$.

We are interested in finding an assignment of users to servers which maximises the sum of the throughputs of all the servers while satisfying the above constraints. Let $x_{m,k}$ denote the number of class k users assigned to server m, $k \in \mathcal{K}$, $m \in \mathcal{M}$, $x_m = \sum_{k=1}^{K} x_{m,k}$, $m \in \mathcal{M}$, and $y_k = \sum_{m \in \mathcal{M}} x_{m,k}$, $k \in \mathcal{K}$. Formally, this problem is

$$\text{maximize} \quad \sum_{m \in \mathcal{M}} \mu_m(x_m) \qquad (P)$$

subject to the constraints

$$\begin{array}{rcll} x_m & \leq & N_m & m \in \mathcal{M} \\ y_k & = & L_k & k \in \mathcal{K} \\ x_{m,k} & \in & \{0, 1, \cdots, N_{m,k}\} & k \in \mathcal{K};\ m \in \mathcal{M} \end{array} \qquad (1)$$

Numerous algorithms [6,7,8,5] exist for solving this problem when there are no class dependent population constraints (i.e., $N_{m,k} = \infty$, $k \in \mathcal{K}$).

Before we describe the algorithm that produces an optimum assignment to this problem, we introduce an *auxiliary* optimization problem P_i, $1 \leq i \leq L$. The solution algorithm for (P) sequentially solves problems P_i, $i = 1, \cdots, L$ where problem P_i is described below. Problem P_L will correspond to problem P.

$$\text{maximize} \quad \sum_{m \in \mathcal{M}} \mu_m(x_m(i)) \qquad (P_i)$$

subject to

$$\begin{array}{rcll} \sum_{m \in \mathcal{M}} x_m(i) & = & i & \\ x_m(i) & \leq & N_m & m \in \mathcal{M} \\ y_k(i) & \leq & L_k & k \in \mathcal{K} \\ x_{m,k}(i) & \in & \{0, 1, \cdots, N_{m,k}\} & k \in \mathcal{K};\ m \in \mathcal{M} \end{array} \qquad (2)$$

where $x_m(i) = \sum_{k=1}^{K} x_{m,k}(i)$, $m \in \mathcal{M}$, and $y_k(i) = \sum_{m \in \mathcal{M}} x_{m,k}(i)$, $k \in \mathcal{K}$. One observes that this problem is identical to the original problem (P) when $i = L$.

3 The Optimal Algorithm

In this section we present an algorithm which produces an optimum assignment to problem P. As mentioned in the last section, solving problem P corresponds to solving problem P_L. Consequently, the algorithm that we present is one that solves problem P_i, $i = 1, 2, \ldots, L$.

3.1 Definitions and Basic Concepts

We call $\mathbf{X} = [x_{m,k}]$ an *assignment* for problem P if and only if

$$\begin{array}{rcll} y_k & \leq & L_k & k \in \mathcal{K} \\ x_{m,k} & \in & \{0, 1, \cdots, N_{m,k}\} & k \in \mathcal{K};\ m \in \mathcal{M}. \end{array}$$

Note that we do not require either that all users be assigned to servers or that the population constraints be satisfied in order to have an assignment. We call $\mathbf{X}$ a *feasible assignment* for problem P if and only if $\mathbf{X}$ is an assignment for problem P and

$$x_m \leq N_m, \quad m \in \mathcal{M}.$$

If all users have not been assigned to servers, then $\mathbf{X}$ is referred to as a *partial* feasible assignment. If all users are assigned to a server (i.e., constraints (1) are satisfied), $\mathbf{X}$ is referred to as a *complete* feasible assignment. When given a feasible assignment $\mathbf{X}$ we will find it useful to consider all unassigned users as residing at a *dummy* server labelled 0, i.e., $x_{0,k} = L_k - y_k$, $k \in \mathcal{K}$

and $x_0 = \sum_{k=1}^{K} x_{0,k}$. We will present an algorithm that operates on a digraph whose nodes are the set $\mathcal{M} \cup \{0\}$.

Definition 1 *If $\mathbf{X}$ is a feasible assignment then $G(\mathbf{X})$ is a digraph with a set of nodes $V = \{0\} \bigcup \mathcal{M}$ and set of edges E where $(m_1, m_2) \in E$ iff $m_1 \in V$, $m_2 \in \mathcal{M}$ and there exists at least one class k such that $x_{m_1,k} > 0$, $N_{m_2,k} > x_{m_2,k}$.*

$G(\mathbf{X})$ represents all potential transfers of a single user from one server to another that are permitted with assignment $\mathbf{X}$. Note that there are no edges directed into the dummy server 0. Also note that if a path exists from server m_1 to m_2 in $G(\mathbf{X})$, then it is possible to produce a new assignment $\mathbf{X}'$ from $\mathbf{X}$ which differs from $\mathbf{X}$ only in the number of users assigned to servers m_1 and m_2. The new assignment will have one less user at server m_1 and one more user at server m_2. If $x_{m_2} < N_{m_2}$ then the resulting assignment will also be feasible. Formally, let $W = (m_1, \ldots, m_n)$ be a path in $G(\mathbf{X})$ iff $(m_i, m_{i+1}) \in E$ for $i = 1, \ldots, n-1$.

Definition 2 *If $\mathbf{X}$ is an assignment to problem (P) and $W = (m_1, \cdots, m_n)$ is an acyclic path in $G(\mathbf{X})$, then $T(\mathbf{X}, W)$ is defined as*

$$T(\mathbf{X}, (m_1, \cdots, m_n)) = T(T(\mathbf{X}, (m_1, m_2)), (m_2, \cdots, m_n)),$$

$n > 2$, where $\mathbf{X}' = T(\mathbf{X}, (m_1, m_2)$ is the following assignment

$$x'_{m,k} = \begin{cases} x_{m_2,k} + 1, & m = m_2; \\ & k = \min\{j | x_{m_1,j} > 0 \wedge x_{m_2,j} < N_{m_2,j}\}, \\ x_{m_1,k} - 1, & m = m_1; \\ & k = \min\{j | x_{m_1,j} > 0 \wedge x_{m_2,j} < N_{m_2,j}\}, \\ x_{m,k}, & \textit{otherwise}. \end{cases}$$

It is important to observe that if $\mathbf{X}$ is a feasible assignment and $x_{m_2} < N_{m_2}$, then $T(\mathbf{X}, W)$ is also a feasible assignment.

Last, let $\Delta_m(x) = \mu_m(x) - \mu_m(x-1)$, $x = 1, 2, \ldots; m \in \mathcal{M}$ denote the incremental change in throughput due to an increase of one user from $x-1$ to x at the m^{th} server.

3.2 Statement of Algorithm

The solution algorithm for P_i assumes that an optimal assignment $\mathbf{X}(i-1)$ has already been obtained for problem P_{i-1}. The algorithm is

0. $\mathcal{M}_a := \{m \in \mathcal{M} \mid N_m > x_m\}$;
1. find $m \in \mathcal{M}_a$ such that $\Delta_m(x_m(i-1)+1)$ is maximized
   if there exists a (acyclic path W from 0 to m in $G(\mathbf{X}(i-1))$
   then $\mathbf{X}(i) := T(\mathbf{X}(i-1), W)$
   else $\mathcal{M}_a := \mathcal{M}_a - \{m\}$;
   if $\mathcal{M}_a := \emptyset$ then halt else goto 1.

A complete solution of P consists of recursively solving P_i for $i = 1, 2, \cdots, L$. The proof of correctness of such a procedure is given in the next section. We shall also determine that if server m is removed from $\mathcal{M}'$ in P_i, then server m will never be used in P_j, $j = i+1, \cdots, L$. A statement of the complete algorithm including the optimisation suggested by the above property is found in Figure 1.

/initialisation/

0. $\mathcal{M}_a := \mathcal{M}$;
1. for $k := 1$ to K do
   $x_{0,k} := L_k$
   for $m \in \mathcal{M}$ do
   $x_{m,k}(0) := 0$;
2. Initialise graph $G(\mathbf{X}(0)$;
   $i := 0$;
3. for $i := 1$ to L do

/solution of P_i/

4. find $m \in \mathcal{M}_a$ such that $\Delta_m(x_m(i-1)+1)$ is maximised;
   if there exists a (acyclic) path W from 0 to m in $G(\mathbf{X}(i-1))$
   then $\mathbf{X}(i) := T(\mathbf{X}(i-1), W)$;
   if $x_m = N_m$ then $\mathcal{M}_a := \mathcal{M}_a - \{m\}$
   else $\mathcal{M}_a := \mathcal{M}_a - \{m\}$
   if $\mathcal{M}_a = \emptyset$ then halt else goto 4

Figure 1: Solution Algorithm.

The function $T(\mathbf{X}, W)$ performs an additional function in the solution algorithm beyond what was stated in its definition. It has the side effect of modifying $G(\mathbf{X})$ to produce $G(T(\mathbf{X}, W))$. One observes that when there are no population constraints, the above algorithm reduces to the greedy algorithm presented in [6].

4 Proof of Correctness of Algorithm

In this section we show that the basic algorithm described in the previous section works correctly. Once this has been established, we then show that at step i if a server is not used, it will never be used later. This establishes the validity of the optimised algorithm presented at the end of the

last section. The proof of correctness of the basic algorithm will follow from a series of three Lemmas.

Lemma 1 *Given a set of servers $\mathcal{M}$, K classes of users with L_k users in the k^{th} class, $k \in \mathcal{K}$, two feasible assignments $\mathbf{X}$ and $\mathbf{X}'$ such that $x'_{m'} = x_{m'} + 1$ for some $m' \in \mathcal{M}$ and $x'_m = x_m$ for $m \neq m'$, then there exists an acyclic directed path from 0 to m' in $G(\mathbf{X})$.*

Proof: We present an algorithm that can be used to construct such a path labeled W. We assume that all users not allocated to servers in $\mathcal{M}$ are placed on dummy server 0. Define $y_k = \sum_{m \in \mathcal{M}} x_{m,k}$ and $y'_k = \sum_{m \in \mathcal{M}} x'_{m,k}$.

0. $W := ()$ { initialise}
1. $W := W \circ (0)$ {add server 0 to the path}
   Find $1 \leq k \leq K$ such that $y'_k > y_k$
   Find $m \in \mathcal{M}$ such that $x'_{m,k} > x_{m,k}$
   $x_{m,k} := x_{m,k} + 1$; $x_{0,k} := x_{0,k} - 1$;
   {transfer class k job from 0 to m}
2. $W := W \circ (m)$ {add m to path }
   if $m = m'$ then halt algorithm
   $n := m$
   Find a $k \in \mathcal{K}$ such that $x'_{n,k} < x_{n,k}$
   $x_{n,k} := x_{n,k} - 1$
   Find an $m \in \mathcal{M}$ such that $x'_{m,k} > x_{m,k}$
   if no such m exists,
   then $x_{0,k} := x_{0,k} + 1$; goto 1
   else $x_{m,k} := x_{m,k} + 1$ goto 2

This algorithm exhibits the following properties:

1. A user is transferred at most one time.
2. The number of transfers performed by the algorithm is bounded from above by $\sum_{m \in \mathcal{M} \cup \{0\}} \sum_{k=1}^{K} |x_{m,k} - x'_{m,k}| < \infty$.

This algorithm always halts as a consequence of the second property. This algorithm produces a directed path W that represents all of the user transfers. As W may contain a cycle, the next step is to remove all cycles and produce a new acyclic path $W' = (0, n_1, \cdots, n_j)$ with the desired properties. This transformation occurs by replacing all subsequences in W of the form $j, \ldots, j$ with j. This yields an acyclic directed path $W' = (0, n_1, \ldots, n_j)$ where $n_j = m'$.

Associated with each pair (n_i, n_{i+1}) in W' is a user that was transferred from n_i to n_{i+1} in the original path. Because of the first property, this user was present at server n_i before the algorithm was executed. Therefore, the edge (n_i, n_{i+1}) is present in $G(\mathbf{X})$.

QED

Lemma 2 *If $\mathbf{X}^{(1)}$ and $\mathbf{X}^{(2)}$ are feasible assignments such that $C_1 = \{m : x_m^{(1)} > x_m^{(2)}\} \neq \emptyset$, $C_2 = \{m : x_m^{(2)} > x_m^{(1)}\} \neq \emptyset$, then for each $m_2 \in C_2$ there is an $m_1 \in C_1$ such that the assignment $\mathbf{X}^{(3)}$ with $x_m^{(3)} = x_m^{(1)}$, $m \neq m_1, m_2$; $x_{m_1}^{(3)} = x_{m_1}^{(1)} - 1$, and $x_{m_2}^{(3)} = x_{m_2}^{(1)} + 1$ is feasible.*

Proof: For a specific $m_2 \in C_2$, construct a new partial assignment $\mathbf{X}^{(4)}$ with

$$x_m^{(4)} = \begin{cases} x_m^{(1)} + 1, & m = m_2 \\ x_m^{(1)}, & m \notin C_1 \cup \{m_2\} \\ x_m^{(2)}, & m \in C_1. \end{cases}$$

This is a feasible assignment.

Construct the additional assignment $\mathbf{X}^{(5)}$ from $\mathbf{X}^{(1)}$ by removing all users from the servers in C_1. This assignment is also feasible.

According to Lemma 1, there is a path, W, between server 0 (the pool of unallocated users) and m_2 in $G(\mathbf{X}^{(5)})$ which can be used to determine what users to move in order to transform $\mathbf{X}^{(5)}$ into $\mathbf{X}^{(4)}$, i.e., $\mathbf{X}^{(4)} = T(\mathbf{X}^{(5)}, W)$. Because of the method used to construct $\mathbf{X}^{(5)}$, the same users can be transferred in assignment $\mathbf{X}^{(1)}$. Consider the first of the users in this transfer sequence. If this user resides on server $m \in C_1$, then the resulting assignment, $T(\mathbf{X}^{(1)}, W)$ is identically $\mathbf{X}^{(3)}$ and $m_1 = m$. If the first user of the sequence resides on server 0 (unallocated pool), then m_1 can be chosen to be any server in C_1 and a user can be removed from it and placed in the unallocated pool (server 0). In other words, moving any user from any server $m \in C_1$ to server 0 in assignment $T(\mathbf{X}^{(1)}, W)$ yields the assignment $\mathbf{X}^{(3)}$ that we are searching for. In this case $m_1 = m$. Therefore $\mathbf{X}^{(3)}$ is a feasible assignment.

QED

Before stating and proving the last lemma, we introduce the following definition.

Definition 3 *Given two feasible assignments $\mathbf{X}$ and $\mathbf{X}'$ for problem P, then $diff(\mathbf{X}, \mathbf{X}') = \sum_{m \in \mathcal{M}} |x_m - x'_m|$.*

Lemma 3 *There exist optimal assignments $\mathbf{X}(i)$ and $\mathbf{X}(i-1)$ that differ from each other at only one server in the number of users assigned to each server. In other words there exists an m' such that*

$$x_m^{(i)} = \begin{cases} x_m^{(i-1)}, & m \neq m' \\ x_m^{(i-1)} + 1, & m = m'. \end{cases}$$

Proof: Assume that the hypothesis is false. Determine two optimal assignments $\mathbf{X}(i)$ and $\mathbf{X}(i-1)$ for problems P_i and P_{i-1} that minimise $diff(\mathbf{X}(i), \mathbf{X}(i-1))$. This value must

be greater than one for the hypothesis to be false. Define the following three sets of servers:

$$\begin{aligned} C_1 &= \{m : x_m(i) > x_m(i-1)\}, \\ C_2 &= \{m : x_m(i) < x_m(i-1)\}, \\ C_3 &= \{m : x_m(i) = x_m(i-1)\}. \end{aligned}$$

Note that neither C_1 nor C_2 can be empty.

We first want to show that there exists some $m_2 \in C_2$ such that $\Delta_m(x_m(i)) < \Delta_{m_2}(x_{m_2}(i)+1)$ for all $m \in C_1$. Choose $m \in C_1$. Construct a new assignment $\mathbf{X}^{(1)}(i-1)$ by starting with $\mathbf{X}(i)$ and removing a user from any server in C_1 other than m. If $C_1 = \{m\}$ then remove the user from m. The key observation is that $x_m^{(1)}(i-1) > x_m(i-1)$.

The above assignment is feasible but not optimal for problem P_{i-1}. If it were optimal, it would yield a contradiction to the statement that $diff(\mathbf{X}(i-1), \mathbf{X}(i))$ is minimal.

Lemma 2 applied to assignments $\mathbf{X}^{(1)}(i-1)$ and $\mathbf{X}(i-1)$ allows us to obtain yet another feasible assignment $\mathbf{X}^{(2)}(i-1)$ that differs from $\mathbf{X}(i-1)$ in that $x_m^{(2)}(i-1) = x_m(i-1)+1$ and there is some $m_2 \in C_2$ such that $x_{m_2}^{(2)}(i-1) = x_{m_2}(i-1)-1$. The number of users assigned to all other servers remain unchanged. Again, this assignment $\mathbf{X}^{(2)}(i-1)$ cannot be optimal. Therefore, $\Delta_m(x_m(i)) \leq \Delta_m(x_m(i-1)+1) < \Delta_{m_2}(x_{m_2}(i-1)) \leq \Delta_{m_2}(x_{m_2}(i)+1)$. We can do this for all $m \in C_1$. Therefore there exists some $m_2 \in C_2$ such that

$$\Delta_m(x_m(i)) < \Delta_{m_2}(x_{m_2}(i)+1), \; m \in C_1. \qquad (3)$$

The last part of the proof consists of applying Lemma 2 to assignments $\mathbf{X}(i)$ and $\mathbf{X}(i-1)$ to yield a new feasible assignment $\mathbf{X}^{(3)}(i)$ such that $x_{m_2}^{(3)}(i) = x_{m_2}(i)+1$ and $x_{m_1}^{(3)}(i) = x_{m_1}(i)-1$ for some $m_1 \in C_1$. According to relation (1) $\mathbf{X}^{(3)}(i)$ exhibits a higher total throughput than $\mathbf{X}(i)$. This contradicts our choice of $\mathbf{X}(i)$ as the optimal assignment. Therefore $\mathbf{X}(i)$ and $\mathbf{X}(i-1)$ can only differ in the number of users allocated to a server at one server, $m' \in \mathcal{M}$.

QED

These lemmas can now be used to prove the main result in the paper.

Theorem 1 *The assignment $\mathbf{X}(i)$ that results from the execution of the algorithm for problem P_i is optimal, $i = 1, \ldots, L$.*

Proof: The proof is by induction on i.
Basis step. The algorithm works trivially for $i = 1$.
Inductive step. We will assume that the hypothesis is true for $i = 1, \ldots, j < L$ and show that it also holds for $i = j+1$. According to Lemma 3, $\mathbf{X}(i)$ and $\mathbf{X}(i-1)$ differ in the number of users assigned at one server, say m'. According to Lemma 1 we know that at least one acyclic directed path exists in $G(\mathbf{X}(i-1))$ from server 0 to server m'. One of these paths is used by the algorithm. Therefore the algorithm produces an optimal assignment.

QED

We complete this section with a proof of the property that the solution algorithm need not further consider a server m if there is ever an instance of the while block where there is no path from 0 to m in $G(\mathbf{X})$. Consider a variation of the solution algorithm where server m is not removed from $\mathcal{M}_a$ in this case. We state and prove the following theorem regarding this algorithm.

Theorem 2 *Let i_0 be the smallest value of i in the* while *block of the modified algorithm such that server m is chosen and there is no path between 0 and m in $G(\mathbf{X}(i_0-1))$. Then m will never be assigned additional users during the remaining execution of the algorithm.*

Proof: Observe that server m will be chosen as the best candidate server in each step of the while block subsequent

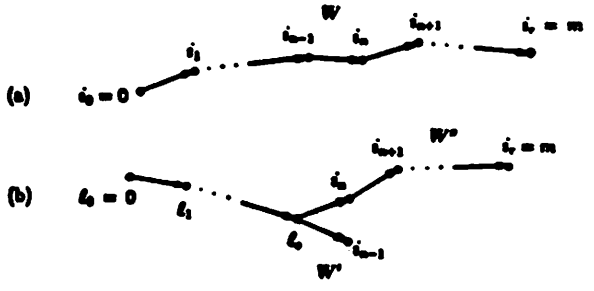

Figure 2: (a) Path from 0 to m in $G(\mathbf{X}(j-1))$. (b) Paths W' and W'' in $G(\mathbf{X}(j-2))$.

to $i = i_0$, at least until server m is assigned an additional user. Assume that the theorem is false and let j be the smallest value of $i > i_0$ such that an additional user is assigned to server m (i.e., $x_m(j) = x_m(j-1)+1 = x_m(i_0-1)+1$). This implies that there is a path $W = (i_0 = 0, i_1, \cdots, i_r = m)$, $(0 < r \leq M)$ from server 0 to server m in $G(\mathbf{X}(j-1))$ (Figure 2(a)) and that no path exists between 0 and m in $G(\mathbf{X}(j-2))$. We will show that this last statement is false.

There is some n such that the path $(i_n, i_{n+1}, \cdots, i_r = m)$, $(0 < n < r)$, exists in $G(\mathbf{X}(j-2))$ and edge (i_{n-1}, i_n) is not present in $G(\mathbf{X}(j-2))$. Furthermore, there must also exist a path $W' = (\ell_0 = 0, \ell_1, \cdots, \ell_s, i_{n-1})$ in $G(\mathbf{X}(j-2))$ (Figure 2(b)). The execution of the while block when $i = j-1$ results in the transfer of jobs within W' including a user from ℓ_s to i_{n-1}. In order for path W to arise, this user is one that can be executed at server i_n. If this is the case,

then this user could also have been transferred from j to i_n during the $(j-1)^{th}$ step. Therefore, there exists a path $W'' = (\ell_1 = 0, \cdots, \ell_s, i_n, i_{n+1}, \cdots, i_r = m)$ in $G(\mathbf{X}(j-2))$ (Figure 2(b)). Consequently, a user can be assigned to m at the $(j-1)^{th}$ step which contradicts the assumption that the earliest assignment occurs at the j^{th} step. QED

As a consequence of this theorem, we can delete a server m for which we are unable to find a path from 0 to m.

5 Computational Requirements of Algorithm

We provide a worst case analysis in this section of the computational requirements of the algorithm described in the section 2. The algorithm requires L executions of the while block. The execution of this block includes

1. determining the maximum of $|\mathcal{M}|$ values,
2. finding a acyclic path from 0 to m (if one exists),
3. computing the new solution $\mathbf{X}(i)$ and $G(\mathbf{X}(i))$.

The first two computations may be executed several times during the execution of the while block. The maximum number of times that they will be executed is $L+M$. The reason for the additional M is because population constraints may require the examination of more than one node before finding one that produces a feasible solution. The third computation is executed at most one time during the execution of the while block. It will be executed a maximum of L times. The first of these computations requires time $O(\log M)$. If the search for a path is performed in a breadth first manner, then the second computation requires time that is $O(M^2)$. In order to determine the complexity of the third item, we provide an algorithm for $T(\mathbf{X}(i-1), W)$ which calculates the new assignment $\mathbf{X}(i)$ and graph $G(\mathbf{X}(i))$. This algorithm differs from the original definition in two respects. As originally stated, T only produced a new assignment. Here it also produces a new graph. Second, it is not defined recursively.

We introduce several parameters. Let $S_k = \{m | N_{m,k} > 0\}$ denote the set of servers that can execute users of class k. Let $E_{i,j} = \{k | i \in S_k \wedge j \in S_k\}$ denote the set of classes that can execute on both server i and j, $i \neq j$, $i,j \in \mathcal{M}$ and let $E_{0,j} = \{k | j \in S_k\}$ denote the set of classes that can execute on server j. Last, let $\mathbf{A}(\mathbf{X}) = [a_{i,j}(\mathbf{X})]$ be an adjacency matrix for the graph $G(\mathbf{X})$ where $a_{i,j}(\mathbf{X}) = |\{k | k \in E_{i,j}, x_{i,k} > 0, x_{j,k} < N_{j,k}\}|$, $i \neq j$ and $a_{i,i} = 0$. Note that when $i \neq j$, $a_{i,j}$ indicates how many different classes of users reside on server i that can be potentially transferred to server j. We shall omit the argument $\mathbf{X}$ when there is no ambiguity regarding its value.

The following algorithm will perform the functions required of $T(\mathbf{X}, W)$. Let $W = (i_1, i_2, \cdots, i_p)$.

```
T(X, W)
 for n = 1 to p − 1 do
   begin
     /step 1/
     find k ∈ E_{i_n,i_{n+1}} such that x_{i_n,k} > 0;
     x_{i_n,k} := x_{i_n,k} − 1; x_{i_{n+1},k} := x_{i_{n+1},k} + 1;
     if N_{i_{n+1},k} = x_{i_{n+1},k} or x_{i_n,k} = 0
       then a_{i_n,i_{n+1}} = a_{i_n,i_{n+1}} − 1;
     /step 2/
     if x_{i_n,k} = 0 then for m ∈ S_k − {i_n, i_{n+1}} do
       if N_{m,k} > x_{m,k} then a_{i_n,m} := a_{i_n,m} − 1;
     /step 3/
     if x_{i_{n+1},k} = N_{i_{n+1},k} then for m ∈ S_k − {i_n, i_{n+1}} do
       if x_{m,k} > 0 then a_{m,i_{n+1}} := a_{m,i_{n+1}} − 1;
     /step 4/
     if x_{i_{n+1},k} = 1 then for m ∈ S_k − {i_{n+1}} do
       a_{i_{n+1},m} := a_{i_{n+1},m} + 1
   end
 end for
```

In the worst case, the number of servers on the path is $O(M)$. Step 1 will require an execution time of $O(K)$, steps 2-4 require execution times that are $O(M)$. Consequently the operation $T(\mathbf{X}, W)$ requires computation that is $O(M(M+K))$.

The computational requirements of the entire algorithm is $O(M(LM + M^2 + LK))$.

6 Example

Consider a set of 5 heterogeneous file servers and a set of 5 classes of workstations. Let the number of workstations belonging to the different classes be given by 50,40,30,20, and 10, respectively. Using a machine/repairman queueing network model, we model a file server and the workstations allocated to it by a single server queue and an infinite server queue, respectively. The response time of the file server corresponds to the repair time, and the elapsed time between the completion of a request to a file server and the generation of the next request corresponds to the interfailure time. The file servers are to be allocated to the workstations so that the total system throughput is maximised subject to the following constraint. Workstations of class $i \in \{1, 2, ..., 5\}$ may be assigned to any of the file servers $(i, i+1, ..., i+I) mod 5$, where $I \in \{0, 1, ..., 4\}$ is a given parameter. This is a model of a one-directional ring network which connects the file servers where workstations of type i are close to file server i. Depending on the value of I, workstations can be assigned to neighboring file servers in one direction only.

This problem maps into our class-constrained resource allocation problem where M=5, K=5, and L_1=50, L_2=40,

L_3=30, L_4=20, and L_5=10, i.e. L=150. For a given $I \in \{0,1,...,4\}$, we have the class constraints

$$N_{m,k}(\mathbf{z}) = \begin{cases} L_k, & m = k,(k+1) mod M,...,(k+I) mod M, \\ 0, & otherwise, \end{cases}$$

and $N_m = L$. The throughput function μ_m is given by

$$\mu_m(\mathbf{z}) = \frac{\gamma \sum_{i=1}^{z} \frac{i(\beta_m/\gamma)^i}{i!}}{\sum_{i=0}^{z} \frac{(\beta_m/\gamma)^i}{i!}}$$

where γ is the failure rate and β is the repair rate in the machine/repairman queueing network model. We let β_1=1, β_2=2, β_3=4, β_4=8, and β_5=16.

For I=0,1,...,4, we use our optimal allocation algorithm to obtain the optimal allocations illustrated in Figure 3. Rows and columns represent classes of workstations and file servers, respectively. From the figure we note that as I increases, the problem becomes less constrained and therefore, we find that the system throughput increases. For a totally constrained system (I=0), the maximum throughput is 10.015, whereas for an unconstrained system (I=4), the maximum throughput is 18.184. It is interesting to observe that even with I=2, where a workstation could be assigned to a local file server or up to only two neighbors to the right (say), the maximum throughput obtained is 17.653, which is close to that of the unconstrained system.

```
ASSIGNMENTS... (I=0)

 50   0   0   0   0
  0  40   0   0   0
  0   0  30   0   0
  0   0   0  20   0
  0   0   0   0  10

 50  40  30  20  10

THROUGHPUT:       10.015
```

```
ASSIGNMENTS... (I=1)

 19  31   0   0   0
  0   0  40   0   0
  0   0   0  30   0
  0   0   0   0  20
  0   0   0   0  10

 19  31  40  30  30

THROUGHPUT:       14.254
```

```
ASSIGNMENTS... (I=2)

  6  14  30   0   0
  0   0   0  40   0
  0   0   0   0  30
  0   0   0   0  20
  0   0   0   0  10

  6  14  30  40  60

THROUGHPUT:       17.653
```

```
ASSIGNMENTS... (I=3)

  0   0  11  39   0
  0   0   0   0  40
  0   0   0   0  30
  0   0   0   0  20
  0   3   3   0   4

  0   3  14  39  94

THROUGHPUT:       18.184
```

```
ASSIGNMENTS... (I=4)

  0   0   0   5  45
  0   0   1  16  23
  0   0   7  10  13
  0   1   4   6   9
  0   2   2   2   4

  0   3  14  39  94

THROUGHPUT:       18.184
```

Figure 3: Optimal Allocations.

7 Conclusions

Several efficient algorithms exist for the single class resource allocation problem. We have extended the problem by considering K classes of L users such that a resource can only be allocated to users belonging to a subset of the various classes. For M resources, we have presented and proved the correctness of an algorithm with worst case time complexity $O(M(LM + M^2 + LK))$ which yields a solution to this class-constrained resource allocation problem that

maximises system throughput. Our algorithm is useful in the optimal allocation of heterogeneous file servers among workstations belonging to different classes under a set of class allocation constraints. It is also useful in the optimal allocation of sessions with different types on heterogeneous host computers with hosts having constraints on the types of sessions they can process.

Various extensions to the problem which we considered are possible. In our model, we assume that the processing requirements of users are class-independent. In other words, the partition of users into classes is used only to restrict the allocation of a resource to a subset of the user classes. A possible extension would consider classes of users with different processing times and routing behavior.

The communications network which connects file servers and workstations is not considered in our model. We assume that the communication time is far less than processing times and access times of direct access storage devices, and therefore is negligible. In some applications, the communication time as well as possible queueing delays due to network congestion may affect overall system performance.

Another area of interest deals with studying throughput concavity, a crucial assumption in our algorithm, in closed queueing networks with different classes of customers and general service times.

References

[1] de Souza e Silva, E. and Gerla, M. "Load Balancing in Distributed Systems with Multiple Classes and Site Constraints," *Proc. Performance'84*, E. Gelenbe, ed., North-Holland, 1984, 17-33.

[2] Dowdy, L. W., Eager, D. L., Gordon, K. D., and Saxton, L.V. "Throughput Concavity and Response Time Convexity," *Information Processing Letters*, vol. 19, (November 1984), 209-212.

[3] Federgruen, A. and Groenevelt, H. "Optimal Flows in Networks with Multiple Sources and Sinks, with Applications to Oil and Gas Lease Investment Programs," *Operations Research*, vol. 34, (March-April 1986), 218-225.

[4] Federgruen, A. and Groenevelt, H. "The Greedy Procedure for Resource Allocation Problems: Necessary and Sufficient Conditions for Optimality," *Operations Research*, vol. 34, (November-December 1986), 909-918.

[5] Frederickson, G. N. and Johnson, D. B. "The Complexity of Selection and Ranking in X+Y and Matrices with Sorted Columns," *J. Computer and System Sci.*, vol. 24 (1982), 197-208.

[6] Fox, B. "Discrete Optimisation via Marginal Analysis," *Management Sci.*, vol. 13, (November 1966), 210-216.

[7] Galil, Z. and Megiddo, N. "A Fast Selection Algorithm and the Problem of Optimum Distribution of Efforts," *JACM*, vol. 26, (January 1979), 58-64.

[8] Katoh, N., Ibaraki, T., and Mine, H. "An Algorithm for the K Best Solutions of the Resource Allocation Problem," *JACM*, vol. 28, (October 1981), 752-764.

[9] Kenevan, J. R. and von Mayrhauser, A. K. "Convexity and Concavity Properties of Analytic Queueing Models for Computer Systems," *Proc. Performance'84*, E. Gelenbe, ed., North-Holland, 1984, 361-375.

[10] Tantawi, A. N. and Towsley, D. "A General Model for Optimal Static Load Balancing in Star Network Configurations," *Proc. Performance'84*, E. Gelenbe, ed., North-Holland, 1984, 277-291.

[11] Tantawi, A. N. and Towsley, D. "Optimal Static Load Balancing in Distributed Computer Systems," *JACM*, vol. 32, 2 (April 1985), 445-465.

[12] Tripathi, S. K. and Woodside, C. M. "A Vertex-Allocation Theorem for Resources in Queueing Networks," to appear in *JACM*.

[13] Woodside, C. M. and Tripathi, S. K. "Optimal Allocation of File Servers in a Local Network Environment," *IEEE Trans. Software Eng.*, vol. 12, 8 (August 1986), 844-848.