RI 10004 06 May 2010 Computer Science

IBM Research Report

Balanced Stream Assignment for Service Facility

Rahul Garg
IBM Research Division
IBM T.J. Watson Research Lab
USA

Perwez Shahabuddin
(formerly) Dept of Industrial Engineering
Columbia University
USA

Akshat Verma
IBM Research Division
IBM India Research Lab
4-Block C, ISID Campus, Vasant Kunj
New Delhi - 110070. India.

Balanced Stream Assignment for Service Facility

Rahul Garg Perwez Shahabuddin Akshat Verma
IBM Research, US IBM Research, India

***Abstract*— Shared data centers and clouds are gaining popularity because of their ability to reduce costs by increasing the utilization of server farms. In a shared server environment, a careful assignment of request streams (e.g., all requests from a customer may constitute a stream) to servers is necessary to ensure good 'end user' performance. In this work, we investigate the assignment of streams to servers in order to minimize an objective function, while ensuring that load is balanced across all the servers. The objective functions we optimize in this work include the overall expected waiting-time, overall probability of the wait exceeding a given value, and stream-weighted versions of these measures.**

We obtain the optimal algorithm for a farm with 2 servers, if sharing of streams among servers is allowed. Based on the insights obtained, we design an efficient algorithm for the multi-server case. By rounding off this solution, we obtain a solution to the case where sharing of streams is not allowed. Our trace-driven evaluation study shows that our algorithms significantly outperform baseline methods. Our work enables high performance for web hosting services as well as emerging Application as a Service (AaaS) clouds. We also show that solutions in areas such as task-level scheduling and file assignment fall within our framework.

I. Introduction

Shared web hosting platforms that allow a large number of small customers to host their web applications have become very prevalent. The unique advantage provided by the shared hosting platforms is their ability to provide high performance to applications at very low cost. Emerging Application as a Service (AaaS) clouds are the next generation of shared data centers that provide different popular application services on a shared application cloud. Google App Engine [10] is a popular application cloud for hosting web applications. Amazon Elastic Compute Cloud (EC2) is a popular Infrastructure as a Service (IaaS) cloud platform. However, even EC2 allows use of many application services like IBM DB2 or IBM Websphere to host customer applications [1].

The benefit of sharing servers on server farms is not just savings in the hardware/software cost (these are slowly becoming commodities), but also to distribute system maintanence costs across several E-businesses. High performance in such a shared hosting platform requires a careful assignment of customers or request streams to servers in the farm. In this work, we look at this problem of deciding which clients (streams) to place on the same server so that some overall performance measure is optimized.

We consider a server farm, where different streams of requests arrive according to the Poisson process with certain arrival rates. Each stream of requests has its own service time distribution with a known mean. For service times with finite variance, the variance is assumed to be known. For some cases of heavy-tailed service times the variance may be infinite; in that case a quantity called the "tail-integral" (defined later) is assumed to be known. Each server is modeled as a single server queue that uses the first-come-first-served (FCFS) service discipline. We address the problem of assigning streams to the servers in order to minimize the overall expected waiting-time, the overall probability that the waiting-time exceeds a certain value (exceedance probability), and stream-weighted versions of these performance measures. The stream-weighted versions of the problem capture differentiated SLA settings, where multiple customers may pay different revenue to the provider. We investigate both the continuous problem (sharing of streams among servers is allowed) and the discrete version. An alternate scenario is when the same server provides different levels of service to different customer classes by using priority schemes and/or generalized processor sharing (GPS; see e.g. [3]). For optimization under this scenario the reader is referred to [17] and references therein.

We can consider a stream to be the aggregated requests from one large client or all clients of a particular service class. There is significant empirical support from many fields that aggregated arrival processes are Poisson processes (see, e.g. , [2], [18]). It is also well known that the rate of such Poisson process are not constant but vary with time of the day, day of the week, etc., i.e., this is what is called a non-homogeneous Poisson process. However, the time scale of change in the rate of the non-homogeneous Poisson process is usually one order of magnitude higher than the time scale of arrival of requests, and one of the ways of modeling such systems is to use piecewise constant rates. Indeed, in our optimization approach one would solve an optimization problem for each length of time that the rates are assumed constant. Alternatively, one may solve the optimization just at the peak rates.

The service-time distribution of request classes are allowed to be arbitrary. An important case that we consider is when they are heavy-tailed. These are random variables whose tail decays at a rate heavier than exponential (e.g., Pareto, lognormal, etc.). The heavy-tails manifest themselves in large variances and probability of exceedances that decay at a sub-exponential rate, e.g., polynomial, in contrast to exponential. There is significant evidence from the World Wide Web that lengths of service times (e.g., session durations), or some quantitities that determine length of service times (e.g., file transfer size) have this property (see, e.g., [5], [16] among many others). The trace service time data that we used for experiments in this paper turned out to lognormal, which is moderately heavy-tailed. Lognormal distributions are also found to fit service times in other related fields like telephone call-centers ([2]).

Finally, we work under the assumption that the load (utilization) on each server is equal. While it is true that the assignment which gives the best overall performance measure may not be the one where the load is balanced at each server, it is usually difficult to convince system administrators to keep the load at different servers unbalanced. Most performance analysis and optimization studies in the task-scheduling area (e.g., [12], [15]) have also relied on this assumption.

A. Contribution

In practice, the naive way to assigns servers to streams, would be to distribute them arbitrarily over the servers, so as to maintain a balanced load (also, called utilization) on servers. The main contribution of the paper is to show that even under the constraint of maintaining balanced load on the servers, one can still improve the overall performance measure by significant amounts, by clever assignment of streams. We present an algorithm that is optimal for a farm with 2 servers. We use this algorithm to design an efficient heuristic for the general case. The efficacy of our proposed algorithm is established using both theoretical and experimental evidence.

Our work also unifies existing work in settings as diverse as task assignment on parallel servers and file assignment problem, which have deterministic service times. We show that these techniques become special case for our proposed algorithm that is designed for stochastic arrival and service times. Our optimization procedure can also be used for server provisioning in web-server farms and AaaS clouds in order to meet certain service level agreements (SLAs), under the scenario that all customer classes with the same service level guarantee are served exclusively by the same set of servers (this seems like a possible, practical scenario). For example, one type of service level guarantee (see, e.g., [17]) specifies that the probability of the waiting-time exceeding a certain value should be below a certain value. Using the optimization procedure presented in this paper, one can minimize this probability for an initial number of servers, and then increase the number of servers and repeat the optimization until the optimal probability falls below the required level.

The rest of the paper is organized as follows. In Section II we define the model and the performance measures, and describe the optimization problem in great generality. In Section III we show how to solve the optimization under the different settings, for the two server case. We use the solution for the two server case as a basis to formulate an algorithm for the general multi-server case. We establish the efficacy of our algorithms using a trace-driven simulation study in Section IV. We extend our model to differentiated QoS and exceedance probability in Sec. V. We present the related work in this area in Section VI and conclude in Sec. VII.

II. Model and Optimization Settings

Consider a queueing system with n arriving request streams (or equivalently, n request classes), and m servers. Each stream, say Stream i, is characterized by $(\lambda_i, E(S_i), E(S_i^2))$ where λ_i is mean arrival rate of stream i and S_i is the random service time. The service time S_i is asssumed to be generally distributed. Each server has its own independent queue and serves requests on a first-come-first-served (FCFS) basis. As mentioned before, arrivals of streams are assumed to be Poisson processes.

The problem we will first try to solve is to find the fraction of each stream to assign to each server in order to:

1) Minimize the overall expected waiting time
2) Minimize the overall $P(\text{Waiting Time} > u)$ for any u.

We consider the cases where all the S_i's are both light-tailed (e.g., exponential, gamma, etc.) and heavy-tailed (these are distributions for which no exponential moments exist, e.g., log-normal, Pareto). The equations for the expected waiting time remains the same in both the light-tailed and heavy-tailed case (as long as $E(S_i^2) < \infty$ for all i in the heavy-tailed case); the heavy-tailedness of the streams reflects itself in higher $E(S_i^2)$'s. However, the equations for the exceedance probability are very different. Remarkably, the structure of the optimization problem in the exceedance probability minimization of the heavy-tailed case, is the same as that for the expected waiting time minimization (more on this later).

Note that in the above formulation, we assume that sharing of streams is allowed, i.e., one can assign fractions of streams to servers. We also consider the case where no sharing of streams is allowed. We will call this the 'discrete' problem, in contrast to the former, which we call the 'continuous' problem. Such discrete optimization problems are usually extremely difficult to solve and NP hard. We find that in the solution to the continuous problem, the number of streams that are shared is of the same order as the number of servers. Hence if the number of streams is much higher than the number of servers, then rounding off the continuous solution usually gives a good solution to the discrete problem. Note that the optimal solution to the continuous problem, is infeasible for the discrete problem, but yields an objective function value that upperbounds the optimal objective value of the discrete problem. The solution we obtain by rounding off the continuous solution is sub-optimal and gives a lower bound to the optimal objective value of the discrete problem. In most experimental examples we saw that the two were very close, indicating that the rounding off technique usually yields good results.

A. Minimizing Expected Waiting Time

Let $\alpha_i^{(j)}$ be the fraction of stream i assigned to server j. Then the total arrival rate to server j is

$$\lambda^{(j)} = \sum_{i=1}^{n} \lambda_i \alpha_i^{(j)}$$

Also, the expected service time and the expected square of the service time faced by server j is given by

$$E(S^{(j)}) = \frac{\sum_{i=1}^{n} \lambda_i \alpha_i^{(j)} E(S_i)}{\lambda^{(j)}} \qquad \text{and}$$

$$E([S^{(j)}]^2) = \frac{\sum_{i=1}^{n} \lambda_i \alpha_i^{(j)} E(S_i^2)}{\lambda^{(j)}},$$

repectively. Then using standard queueing theory (see, e.g., [11]), the expected waiting at node j is given by

$$E(W^{(j)}) = \frac{\lambda^{(j)}E([S^{(j)}]^2)}{1-\lambda^{(j)}E(S^{(j)})} = \frac{\sum_{i=1}^{n}\lambda_i\alpha_i^{(j)}E(S_i^2)}{1-\sum_{i=1}^{n}\lambda_i\alpha_i^{(j)}E(S_i)}$$

and hence the overall expected waiting is given by

$$\begin{aligned} E(W) &= \sum_{j=1}^{m}\left(\frac{\lambda^{(j)}}{\lambda_{total}}\right)E(W^{(j)}) \\ &= \sum_{j=1}^{m}\left(\frac{\sum_{i=1}^{n}\lambda_i\alpha_i^{(j)}}{\sum_{i=1}^{n}\lambda_i}\right)\frac{\sum_{i=1}^{n}\lambda_i\alpha_i^{(j)}E(S_i^2)}{1-\sum_{i=1}^{n}\lambda_i\alpha_i^{(j)}E(S_i)} \end{aligned}$$

Here $\lambda_{total} = \sum_{i=1}^{n}\lambda_i$ is the total arrival rate to the system. Note that $\lambda^{(j)}E(S^{(j)})$ is the load faced by Server j (also known as the utilization, or the traffic intensity). For stability we need $\lambda^{(j)}E(S^{(j)}) = \sum_{i=1}^{n}\lambda_i\alpha_i^{(j)}E(S_i) \leq 1$ for each j. The problem then becomes minimize $E(W)$, subject to the constraints

$$\alpha_i^{(j)} \geq 0 \;\; \forall i,j, \;\; \sum_{j=1}^{m}\alpha_i^{(j)} = 1 \;\; \forall i, \;\; \sum_{i=1}^{n}\lambda_i\alpha_i^{(j)}E(S_i) \leq 1 \;\; \forall j$$

Hence we have an optimization problem where the decision variables are the $\alpha_i^{(j)}$'s. Note that we have convex linear constraints, but a very complicated objective function. The objective function can be shown to non-convex, so most standard non-linear programming packages cannot be used here. Also, the problem has nm variables and $mn+n+m$ constraints.

B. Mapping to a Space with Fewer Dimnesions

Our original problem is an optimization problem with mn dimensions. We now reforumlate the problem to a version that has significantly fewer number of dimensions and is easier to visualize geometrically. We define

- $X^{(j)} = \frac{\sum_{i=1}^{n}\lambda_i\alpha_i^{(j)}}{\lambda_{total}}$
- $Y^{(j)} = \sum_{i=1}^{n}\lambda_i\alpha_i^{(j)}E(S_i^2)$
- $Z^{(j)} = \sum_{i=1}^{n}\lambda_i\alpha_i^{(j)}E(S_i)$

We also make the following simplification in notation: $a_i \equiv \lambda_i/\lambda_{total}$, $b_i \equiv \lambda_i E(S_i^2)$, $c_i \equiv \lambda_i E(S_i)$. We can regard (a_i, b_i, c_i) as the initial data for the optimization problem. Then the optimization problem becomes

$$\text{Minimize} \quad \sum_{j=1}^{m}\frac{X^{(j)}Y^{(j)}}{1-Z^{(j)}} \quad \text{subject to}$$

$$X^{(j)} = \sum_{i=1}^{n}\alpha_i^{(j)}a_i, \;\; Y^{(j)} = \sum_{i=1}^{n}\alpha_i^{(j)}b_i, \;\; Z^{(j)} = \sum_{i=1}^{n}\alpha_i^{(j)}c_i; \;\; \forall j$$

$$\alpha_i^{(j)} \geq 0 \quad \forall i,j, \quad \sum_{j=1}^{m}\alpha_i^{(j)} = 1 \quad \forall i, \quad \sum_{i=1}^{n}\alpha_i^{(j)}c_i \leq 1 \quad \forall j, \tag{1}$$

Define $A \equiv \sum_{i=1}^{n}a_i$, $B \equiv \sum_{i=1}^{n}b_i$, and $C \equiv \sum_{i=1}^{n}c_i$. Note that C is the total load on the system, and we assume $C < m$ for stability.

The usual way of viewing this problem, as suggested by our initial formulation, would be to view it as a minimization problem on the feasible set of the $\alpha_i^{(j)}$'s. Another way as suggested by the later formulation, is to view it as a minimization problem on the feasible set of the $(X^{(j)}, Y^{(j)}, Z^{(j)})$'s (note that only these are present in the objective function). Whereas the former feasible space is in mn dimensions, the feasible space in the latter formulation is just in $3m$ dimensions. The main problem in the latter approach, of course, is to identify this feasible set of the $(X^{(j)}, Y^{(j)}, Z^{(j)})$'s, since they are expressed not in terms of each other, but in terms of "supplementary variables", i.e., the $\alpha_i^{(j)}$'s.

C. Unconstrained and Constrained Formulations

High load on servers increase the probability of failure in a system. Hence, system administrators often prefer to load balance all the servers in their server farm. To capture this aspect of the problem, we add the following constraint

$$Z^{(j)} = C/m \quad \forall j \tag{2}$$

and denote it by *Constrained Problem Optimization*. Typical data centers often exhibit very low average utilization (as low as 10% in a study reported in [20]). We call these phenomenon as light traffic situation and handle it differently. This version of the problem is not only interesting in its own right but also provides an insight to solve the general problem. Recall that C is the total load on the system. The light traffic situation is when $C \approx 0$. Since $\sum_{i=1}^{m}Z^{(j)} = C$, we have that $Z^{(j)} \approx 0$ for all j. Since the balanced load constraint is absent for the light traffic problem, we denote this problem as the *Unconstrained Problem Optimization* and capture is as

$$\text{Minimize} \quad \sum_{j=1}^{m}X^{(j)}Y^{(j)} \quad \text{subject to the constraints of (1).} \tag{3}$$

(the constraint $\sum_{i=1}^{n}\alpha_i^{(j)}c_i \leq 1$ is now redundant).

III. Optimization Methodology

A. Method Overview

In order to solve the problem, we first consider the unconstrained problem for the case where there are only 2 target servers for placing the streams. It is easy to see that the constrained version has the same objective function. However, it needs to satisfy an additional constraint (of load balancing) and may have a much smaller feasible space.

One may note that our objective function is non-convex, which makes our problem difficult to solve. We identify the feasible space and divide into regions. We observe that the optimal falls in one of the regions. Further, we show that the objective function has convex level sets in this region. Hence, our methodology performs a greedy search in this region and terminates when it finds a local optimal. The convex level set property ensures that the local optimal in this region is the global optimal as well, leading to our optimality result for the 2-server unconstrained problem. The objective function for the constrained problem is same as the unconstrained problem and the feasible space is a subset of the unconstrained problem. Hence, the convex level set property is satisfied for this problem as well. We use a stream swapping technique to find a local optimal in this reduced feasible space and use the convex level set property to argue that our solution is also the global optimal. Our 2-server algorithms are then used iteratively to find a heuristic solution for the general m-server scenario.

B. Two Server Case

1) Optimizing the Unconstrained Problem: We start with the case of exactly two servers, and give the exact optimal for this case. Then we extend the algorithm for the m server case.

Consider (3) for the case of two servers. Note that $X^{(1)} + X^{(2)} = \sum_{i=1}^{n} a_i \equiv A$ and $Y^{(1)} + Y^{(2)} = \sum_{i=1}^{n} b_i \equiv B$. Using this fact, (3) reduces to

$$\text{Minimize} \quad X^{(1)}Y^{(1)} + (A - X^{(1)})(B - Y^{(1)}) \quad \text{s.t.}$$
$$X^{(1)} = \sum_{i=1}^{n} \alpha_i^{(1)} a_i, \qquad Y^{(1)} = \sum_{i=1}^{n} \alpha_i^{(1)} b_i$$
$$0 \le \alpha_i^{(1)} \le 1 \quad \forall i \tag{4}$$

So, one can view the above problem as one on two dimensional space (i.e., the space of $(X^{(1)}, Y^{(1)})$). We solve the problem by the following appraoch. We find the feasible region first. We then characterize the objective function in a part of the feasible space that contains the optimal.

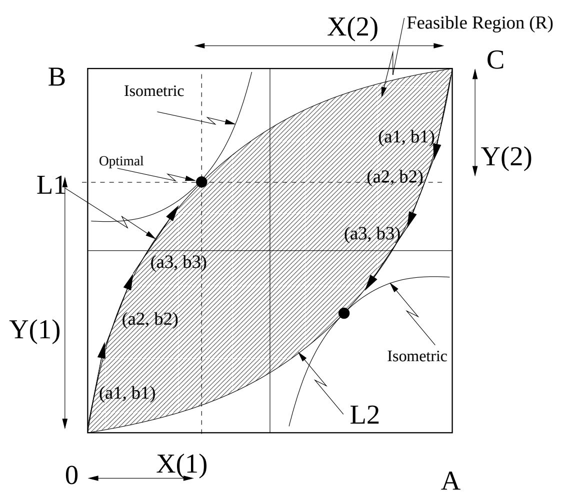


Fig. 1. Feasible region for Unconstrained Optimization

Identifying the feasible region: The following algorithm determines the feasible region.

- Compute the slope b_i/a_i of the vector (a_i, b_i) corresponding to stream i. Note that for case of the expected delay, this ratio is just $E(S_i^2)$, and and for the case of exceedance probability in the heavy-tailed case, this ratio is just $\int_{s=0}^{u}(1 - F_i(s))ds$.
- Order the streams i in the order of decreasing slopes. Without loss of generality, henceforth assume that the streams were ordered in this manner from the beginning.
- Plot the piecewise linear curve L1 joining $(0, 0)$, (a_1, b_1), $(a_1 + a_2, b_1 + b_2)$, $\ldots$, $(\sum_{i=1}^{n-1} a_i,\ \sum_{i=1}^{n-1} b_i)$, (A, B) and curve L2 joining (A, B), $(\sum_{i=1}^{n-1} a_i,\ \sum_{i=1}^{n-1} b_i)$, $\ldots$, (a_1, b_1), $(0, 0)$ as shown in Figure 1.

Lemma 1: The feasible region for this problem is the region R between the two curves $L1$ and $L2$ as shown in Figure 1.

We now show that the objective function has convex level sets in the region $X^{(1)} \le A/2$ and $Y^{(1)} \ge B/2$, and the optimal solution can be be obtained in this region. Therefore the optimal solution lies on the boundary of the restricted feasible region. This can be obtained by moving on the boundary of the region along the direction of improvement of objective function until it cannot be improved. The algorithm to find optimum begins at point $(0, 0)$ and adds streams to the first server in decreasing order of their slopes until $Y^{(1)}$ becomes $B/2$. After this it continues to add streams to the first server until the objective function stops improving.

A useful property of the objective function in the region that contains the optimal: The following lemma can be shown using standard techniques:

Lemma 2: The function $f(x, y) = xy + (A - x)(B - y)$ has convex level-sets in the region $x \le A/2$, $y \ge B/2$.

Note also that the values of the function in the region $x \le A/2$, $y \ge B/2$ is symmetric to those on the region $x \le A/2$, $y \le B/2$ and the region $x \ge A/2$, $y \ge B/2$. Therefore the optimal solution lies on the boundary of the feasible region restricted to the quadrant $x \le A/2$, $y \ge B/2$. This can be obtained by moving on the boundary of the region along the direction of improvement of objective function until it cannot be improved. In particular the algorithm begins at point $(0, 0)$ and adds streams to the first server in decreasing order of their slopes until $Y^{(1)}$ becomes $B/2$. After this it continues to add streams to the first server until the objective function stops improving.

2) Constrained Problem Optimization: Again, we first give an optimal algorithm for the two-server case, and then use it in a heuristic for the m-server case.

Note that now we have to add the balanced load constraint, i.e., $\sum_{i=1}^{n} \alpha_i^{(1)} c_i = C/2$ to the minimization problem given by (3). To simplify notation we will denote $\alpha_i^{(1)}$ by α_i. If we make the change in variable $\tilde{\alpha}_i = \alpha_i c_i$, then the new constraint set becomes:

$$X^{(1)} = \sum_{i=1}^{n} \tilde{\alpha}_i \tilde{a}_i \qquad \text{and} \quad Y^{(1)} = \sum_{i=1}^{n} \tilde{\alpha}_i \tilde{b}_i$$
$$\sum_{i=1}^{n} \tilde{\alpha}_i = C/2 \quad \text{and} \qquad 0 \le \tilde{\alpha}_i \le c_i \quad \forall i \tag{5}$$

where $\tilde{a}_i = a_i/c_i$ and $\tilde{b}_i = b_i/c_i$. Let $\mathbf{v}_i$ be the vector $(\tilde{a}_i, \tilde{b}_i)$. Let $\mathbf{w}_{ij} = v_i - v_j$.

For simplicity, we will assume that the $(\tilde{a}_i, \tilde{b}_i, c_i)$'s are sufficiently arbitrary, so that

Assumption 1: (a) No combinations of c_i's will sum up to exactly $C/2$, i.e., all feasible stream allocations need to share at least one stream between servers. (b) No two vectors in the set $\mathcal{V} = \{\mathbf{v}_i : i \in (1, \ldots, n)\}$ have the same direction. (c) No two vectors in the set $\mathcal{W} = \{\mathbf{w}_{ij} : i \in (1, \ldots, n), j \in (1, \ldots, n), i \neq j\}$ have the same direction.

The addition of the balanced load constraint makes it much

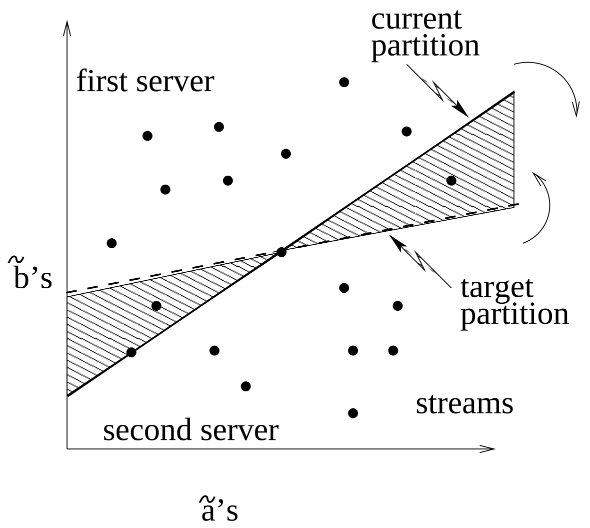


Fig. 2. Finding optima in two server case of Constrained Optimization

more difficult to identify the *new* feasible region R in the $(X^{(1)}, Y^{(1)})$ space. Of course, we know that it is subset of the R in the light-traffic case, and that it has piecewise-linear boundaries. Instead of determining the feasible region explicitly as in the light-traffic case, we adapt an approach based on a (continuous) exchange of streams between the two servers such that feasibility is always preserved (i.e., the load on each server remains equal). The problem at each step is to select two streams (one on each server) where an exchange of these two streams between the two servers results in decrease in the objective function. Of course, we start with an initial solution that satisfies the balanced load constraint and is trivial to find (see later). We have identified the following property that helps us define swapping any pair of streams between a pair of servers reduces the objective function or not.

Definition 1: Define two servers j and l to be satisfying the exchange property if $\exists i, k$ such that $\alpha_i^{(j)} > 0, \alpha_k^{(l)} > 0$ and

$$(\tilde{a}_k - \tilde{a}_i)(Y^{(j)} - Y^{(l)}) + (\tilde{b}_k - \tilde{b}_i)(X^{(j)} - X^{(l)}) < 0.$$

We have proved the following result on the exchange property that helps us terminate our procedure.

Theorem 1: A feasible solution $\tilde{\alpha}$ is a local minima, if any only if the two servers do not satisfy the exchange property.

Since the objective function in the two server case has convex level sets in the desired region, the local minima is also its global minima. In order to find a optimal solution, we first find an initial feasible solution and then refine this solution successively using the exchange property, until the property is no longer satisfied.

Represent the stream vectors $\mathbf{v}_i$'s as points in in the first quadrant of the XY plane as shown in Figure 2. At every stage the algorithm maintains a feasible solution by partitioning the set of streams between the two servers using a straight line. All the streams strictly above the line are assigned to the first server ($\tilde{\alpha}_i = c_i$) and the streams strictly below the line are assigned to the second server ($\tilde{\alpha}_i = 0$). The streams on the line are divided between the two servers.

The initial feasible solution is found using a line through the origin with a slope β such that:

$$\sum_{i:\tilde{b}_i/\tilde{a}_i < \beta} c_i \leq C/2 \text{ and } \sum_{i:\tilde{b}_i/\tilde{a}_i > \beta} c_i \leq C/2$$

Such a line can be made to pass through a point, say k, and $\tilde{\alpha}_k$ can be appropriately assigned such that:

$$\sum_{i:\tilde{b}_i/\tilde{a}_i > \beta} c_i + \tilde{\alpha}_k = C/2$$

Now, the algorithm maintains two partitions: (i) a current partition that represents the current feasible solution and (ii) a target partition that represents the direction in which the current partition should be modified to obtain a better feasible solution. At any stage, the algorithm also keeps a tab on the stream (say k) that is shared between the two servers.

Without loss of generality, assume that $X^{(2)} > X^{(1)}$. The target partition is defined by the line of slope γ passing through the point k, where γ is given by:

$$\gamma = (Y^{(2)} - Y^{(1)})/(X^{(1)} - X^{(2)})$$

The next two theorems establish the condition for optimality.

Theorem 2: If the current partition is same as the target partition then the solution is optimal.

Theorem 3: If current partition is not the same as the target partition, then the solution is not optimal.

For the case of deterministic service times, note that $E(S_i^2) = E^2(S_i)$ and hence one sorts the streams based on decreasing $E(S_i)$. In that case, $b'_i = E(S_i)$ is a decreasing sequence and $a'_i = 1/E(S_i)$ is an increasing sequence. One can then check that the initial partition is the same as the target partition, and is thus optimal, thus giving support to the result in [15].

We now present an algorithm to find the optimum solution in two server case. Without loss of generality assume that $\beta > \gamma$. The algorithm finds a point i in the shaded region (Figure 2) that maximizes the slope of the vector $\mathbf{v}_i - \mathbf{v}_k$. Formally, it chooses i such that $\tilde{\alpha}_i = 0$ and $\tilde{b}_i - \tilde{b}_k < \beta(\tilde{a}_i - \tilde{a}_k)$ and $(\tilde{b}_i - \tilde{b}_k)/(\tilde{a}_i - \tilde{a}_k)$ is maximized.

It then rotates the line defining the current partition clockwise until either, one of the points i and k is no longer shared between the two servers, or the current partition becomes same as the target partition. Formally, let

$$\delta = \min\left(c_i, \tilde{\alpha}_k, \frac{1}{4}\left(\frac{Y^{(2)} - Y^{(1)}}{b_i - b_k} + \frac{X^{(2)} - X^{(k)}}{a_i - a_k}\right)\right)$$

The quantity δ is added to $\tilde{\alpha}_i$ and removed from $\tilde{\alpha}_k$. In case, the current partition is still different from the target partition, the new shared stream becomes i if $\delta = c_k - \tilde{\alpha}_k$ and k if $\delta = c_i$. The process is repeated with the new shared stream. This continues till the target partition becomes same as the current partition. Figure 3 gives the formal details of the algorithm.

Theorem 4: The algorithm terminates in $O(n^2 \log n)$ steps.

C. Multi Server Case

1) The Unconstrained Problem: Now consider the case with $m > 2$ servers. We first graphically represent

algorithm SWEEP
  $Y = \sum_{i=1}^{n} \tilde{b}_i; X = \sum_{i=1}^{n} \tilde{a}_i$
  Find β, k and $\tilde{\alpha}_k$ such that:
    $\sum_{i:\tilde{b}_i/\tilde{a}_i>\beta} c_i + \tilde{\alpha}_k = C/2$
  $Y^{(1)} = \sum_{i:\tilde{b}_i/\tilde{a}_i>\beta} \tilde{b}_i + \tilde{\alpha}_k \tilde{b}_k; Y^{(2)} = Y - Y^{(1)}$
  $X^{(1)} = \sum_{i:\tilde{b}_i/\tilde{a}_i>\beta} \tilde{a}_i + \tilde{\alpha}_k \tilde{a}_k; X^{(2)} = X - X^{(1)}$
  $\gamma = (Y^{(2)} - Y^{(1)})/(X^{(1)} - X^{(2)})$
  while $\gamma < \beta$
    $i = \arg\max_{l:\tilde{b}_l/\tilde{a}_l<\beta} (\tilde{b}_l/\tilde{a}_l)$
    $\delta = \min\left(c_i, \tilde{\alpha}_k, \frac{1}{4}\left(\frac{Y^{(2)}-Y^{(1)}}{\tilde{b}_i-\tilde{b}_k} + \frac{X^{(2)}-X^{(k)}}{\tilde{a}_i-\tilde{a}_k}\right)\right)$
    $\tilde{\alpha}_i = \tilde{\alpha}_i + \delta; \tilde{\alpha}_k = \tilde{\alpha}_k - \delta$
    $X^{(1)} = X^{(1)} + \delta\tilde{a}_i - \delta\tilde{a}_k; X^{(2)} = X - X^{(1)}$
    $Y^{(1)} = Y^{(1)} + \delta\tilde{b}_i - \delta\tilde{b}_k; Y^{(2)} = Y - Y^{(1)}$
    $\gamma = (Y^{(2)} - Y^{(1)})/(X^{(1)} - X^{(2)})$
    $\beta = (\tilde{b}_i - \tilde{b}_k)/(\tilde{a}_i - \tilde{a}_k)$
    if $\tilde{\alpha}_k = 0$ then $k = i$
    else if $\beta = \gamma$ then done
  end while
end algorithm SWEEP

Fig. 3. Algorithm SWEEP for finding optima in balanced load two server case

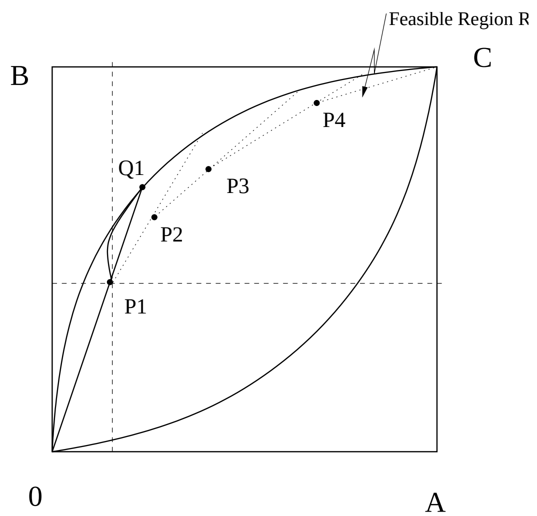


Fig. 4. Feasible region for light traffic: Multiple Servers

the feasible region and feasible solutions. In any solution, order the servers such that $Y^{(1)}/X^{(1)} \geq Y^{(2)}/X^{(2)} \ldots Y^{(m)}/X^{(m)}$. Any solution can be represented as a sequence of $m-1$ points $(X^{(1)}, Y^{(1)})$, $(X^{(1)} + X^{(2)}, Y^{(1)} + Y^{(2)})$, $\ldots (\sum_{j=1}^{m-1} X^{(j)}, \sum_{j=1}^{m-1} Y^{(j)})$.

Lemma 5: A solution is feasible iff all the corresponding $m-1$ points lie inside the region R.

A geometric proof of this lemma is presented in Fig. 4 and the detailed proof is available in the appendix. The proof is by induction and works by taking an assignment to m servers, removing the streams assigned to the first server and reducing it to a problem with $m-1$ servers (the region starting from $P1$ in Fig. 4). We state the following intuitive result next (proof in appendix).

Lemma 6: The optimal solution will have all the $m-1$ points on the boundary of the region R.

The heuristic starts with a initial feasible solution where the $m-1$ points are placed equally spaced on the boundary of the feasible region. Then the algorithm successively improves this solution by considering adjacent servers and optimizing the stream allocation locally within these two servers. It begins with optimizing the first two servers, then second and third server and so on, till the $m-1^{th}$ and m^{th} server. Each such pass of the optimization improves the objective function value and gives another solution on the boundary. These passes are repeated until the improvement in the objective function value is zero or insignificant (in our experiments, we define insignificant as being less than 1 %).

Lemma 7: If feasible solution cannot be improved locally using *any* pair of two adjacent servers, then it is a local optima in the feasible region.

In the 2-server case, sorting the stream takes $O(n \log n)$ time, whereas assigning the stream takes $O(n)$ time. Hence the total running time is $O(n \log n)$. For the m-server case, the running time is $O(k \sum_{j=1}^{m} n_j log n_j) \leq O(knlogn)$ where k is the number of passes and n_j is the number of streams allocated to server j and $j+1$. In all our experiments, k turned out to be less than 5.

2) Constrained Problem Optimization: We use the two-server optimal algorithm in a heuristic for the general m-server problem. As before, the algorithm begins by sorting the streams based on decreasing $\tilde{b}_i/\tilde{a}_i$. It then allocates streams to the first server until the workload on it reaches C/m (typically, the last stream allocation to the server is a fractional one). This is done sequentially on the other servers, so that in the end each server has workload C/m. This constructs an initial feasible allocation. Then, the two-server algorithm is applied on the first two servers. Once the two-server algorithm returns a new allocation for server j and $j+1$, it is applied again on server $j+1$ and $j+2$ until $j+2 = n$. In this way, one pass of the heuristic is completed. The algorithm now traces back from n to 1 for a second pass. The procedure can be repeated until any further application of the procedure does not lead to any significant improvement in the objective function.

The worst case running time of the above algorithm is $O(k \sum_{j=1}^{m} n_j^2 \log n_j)$ where k is the number of passes, n_j is the total number of streams shared between servers j and $j+1$. In the worst case, the running time could be $O(kn^2 \log n)$. However, in our experiments, k was always less than 5 and the total number of steps taken by **SWEEP** in each pass of the algorithm never went beyond $10n$ even for n as large as 1500. Hence the actual running time turned out to be almost linear for large values of n.

IV. Experimental Results

We conducted experiments to study the effectiveness of our algorithms against competing methodologies that are likely to be used in practice.

A. Experimental Procedure

We tried our algorithm on different sets of traces obtained from different webservers. The traces were obtained from the Internet Traffic Archive. Each trace consists of a group of IP addresses that send a request stream to a particular web-server. We consider requests from each IP address to be a customer class or equivalently, a stream, and denote it as before by i. For each IP address, the absolute times of the requests arrival at the server, and the data transfer sizes when the requests were made, was recorded. We assume that the sevice time for a request is proportional to the data transfer size. Hence we use the sequence of data tranfer sizes as our S_i's and estimate the $E(S_i)$ and $E(S_i^2)$, by the sample mean of the S_i's and S_i^2's, respectively. We checked for autocorrelation of various lags in the sequence of service times and found them to be at most 0.2, which is small, even though not negligible. For simplicity we assume these sequences to be un-autocorrelated. The absolute times of the request arrivals was used to estimate λ_i, the arrival rate of requests from that IP address. After estimating the λ_i, $E(S_i)$, and $E(S_i^2)$, we then calculated the data of our problem, i.e., the a_i, b_i, c_i values that were defined before. We used two traces that were of different nature. The first trace is a day's worth of all HTTP requests to the EPA WWW server located at Research Triangle Park, NC [7]. The second trace contains all the HTTP requests made to the Clarknet WWW server (Clarknet is a full Internet access provider for the Metro Baltimore-Washington DC area) for one week [4]. We picked up requests from 900 IP addresses from the first trace and call it Set A. Similarly, we picked up the requests from 1500 IP addresses from the second trace and labelled it as Set B.

B. Unconstrained Optimization Results

In order to evaluate our algorithm for *Unconstrained Optimization*, we also implemented the following baseline algorithms and performed a comparative study.

Random Pick Algorithm (RPA): For each stream, the algorithm picks one of the servers randomly.

Minimum Delay Algorithm (MDA): For each stream, it picks the server which has the least current (i.e., using all the streams picked so far) performance measure value (i.e., $E(W^j)$ in the case of expected delay minimization). Ties are resolved randomly.

For the light traffic case, the average delays reported are relative, as we do not take the load factor into consideration. However, the results are useful for a comparative study of the various algorithms, which is the aim of these experiments. In order to get the integral solution from the continous solution, we simply move a fragmented stream to the server that has the higher Y_i/X_i ratio.

We note that our algorithm comprehensively outperforms the other algorithms considered (Fig. 5). Another important observation we would like to make is the fact that the integer case and the continous case report very similar numbers. Hence, even a trivial rounding technique leads to an integral solution that has an objective balue very close to that of the continous solution. Moreover, the performance improvement is significant, reaching even an order of 10 in some cases. We also note that the performance improvement is fairly consistent with varying number of servers as well. Recall that in the 2-server case, our algorithm is provably optimal, whereas, in the k-server case, our algorithm is only a heuristic. However, the performance improvement, when we increase the number of servers from 2, does not decrease. This leads one to believe that our algorithm terminates fairly close to the global optimal, even for $k > 2$.

C. Constrained Optimization Results

We now present an evaluation of our proposed algorithm for *Constrained Optimization*. We again compare the performance of our proposed algorithm with the follwoing baseline algorithms. The baseline algorithm also ensure that each server gets approximately equal load in the end.

Minimum Delay Algorithm (MMDA): For each stream, it picks the server which has the least current performance measure value (i.e., $E(W^j)$ in the case of expected delay minimization) using all the streams picked so far. Ties are resolved randomly. Servers whose load reaches or exceeds C/m are not considered in further iterations.

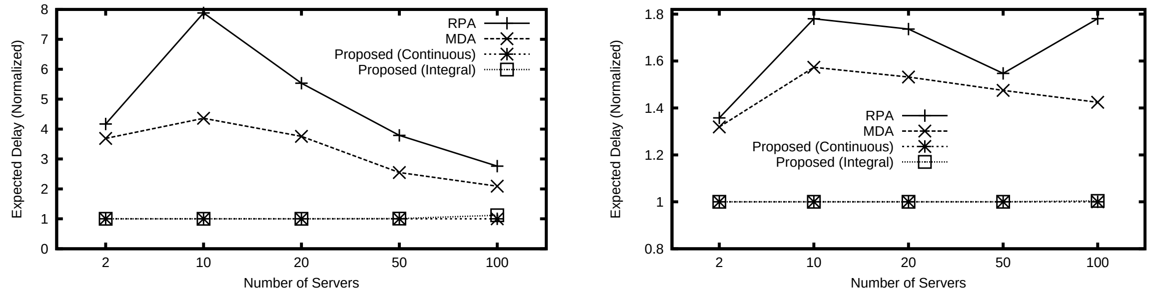
Minimum Load Algorithm (MLA): The algorithm assigns streams to servers taking one stream at a time. It sends each stream to a server which has the least $Z^{(j)}$ value (using the streams assigned so far). Note that in the end, each server will roughly have the same load.

Fill Algorithm (FA): The algorithm takes one stream at a time in an arbitrary order. It assigns streams to a server until the workload $Z^{(j)}$ reaches C/m. After the threshold is reached, it moves on to the next server.

Equal Partitioning Algorithm (EPA): The algorithm partitions each stream into equal parts and assigns one partition to each server. Note that even in this case, each server will get a load of exactly C/m.

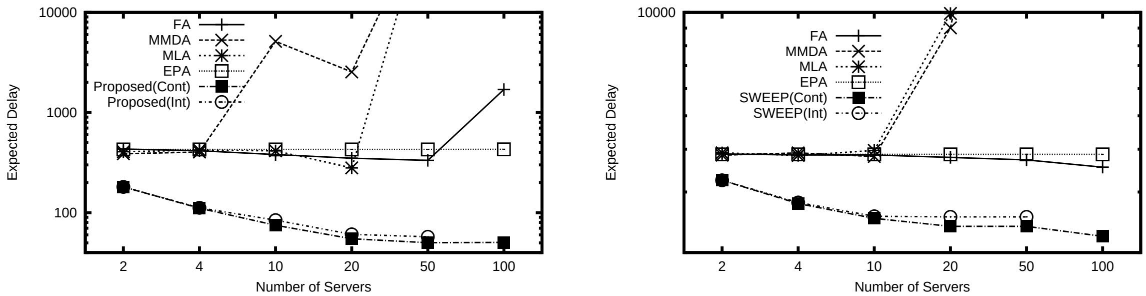
We conducted experiments to study the performance of our algorithms vis-a-vis the above algorithms under a load factor C/m of 0.9. We varied the number of servers from 2 to 100 to study the performance with varying number of servers. The results (Fig. 6) show the superiority of our algorithm against any of the competing algorithms, i.e., our algorithm shows a performance improvement upto a factor of 10 over the competing algorithms. The performance improvement is significant for both the web-traces.

We also note that the integral solution is better than any other comparative method for all cases other than the 100 server case, even though some of the other methods report a non-integral solution. Also, when the number of servers are small, the performance loss in moving from a continous solution to an integral one is very small. However, this penalty becomes significant as the number of servers is increased. This is not because of any scaling problem that the algorithm may have but because of the fact that an increase in the number of servers leads to a small number of sessions being allocated to each server. As a result, moving even a single session can change the load on a server significantly that in turn may also change the objective function. One may also note that the *MMDA* and the *MLA* algorithm does not even return a feasible solution in such a scenario. This essentially leads one to believe that the integral solutions are very far off from their continous counterparts when the number of sessions per server is low. This is not surprising because in such a scenario, addition of a single session can increase the workload to close to 1 or even beyond 1, as was also seen in the case of *MMDA* and *MLA*. However, typical servers often serve multiple customer sessions and a low session per server scenario can be ignored for all practical purposes. As one may note, when the number of sessions per server is high,



(a) (b)

Fig. 5. Normalized Weighted Expected Delay for (a) Data Set A and (b) Data Set B for Unconstrained Optimization



(a) (b)

Fig. 6. Normalized Weighted Expected Delay for (a) Data Set A and (b) Data Set B for Constrained Optimization

our integral solution is very close to the continous case and comfortably outperforms all other algorithms, both continous and integral.

Note that in the 2-server case, our algorithms yield the exact optimal solution and the performance is the best possible. However, even for the m server case ($m > 2$), we see a fairly marked performance improvement. This suggests that even when $m > 2$, our algorithms are fairly close to the optimal solution.

Convergence Speed: We also estimated the running time of our proposed algorithm. We observed that the method **SWEEP** was never called for more than 10 steps in the 2-server sub-optimizations, and we never needed to run more than 5 passes. Another interesting fact was that the initial solution constructed by our algorithm was also very close to the optimal. We never observed it to be more than 10% away from the final solution. This leads us to an important insight that even sorting the streams based on b_j/a_j and allocating streams gives a very good solution as compared to the naive methods. The simplicity of the initial solution makes it a good choice for system managers who are not aiming for an optimal solution and would sacrifice a bit of performance for simplicity.

V. Model Extensions

A. Minimimizing Exceedance Probability

We now look at the related problem of minimizing $P(W > u)$ for some u. Of course, in queueing theory no exact results exist for the waiting time of $M/G/1$ server, so we make use of a heavy-traffic approximation. The term heavy traffic means that the load of each server, i.e., $Z^{(j)} = \lambda^{(j)} E(S^{(j)})$ is close to 1. For any server j, heavy traffic approximation (valid also for non-Poisson arrivals) says that

$$P(W^{(j)} > u) \approx \exp\left\{-\frac{[1-\lambda^{(j)}E(S^{(j)})]u}{\lambda^{(j)}E([S^{(j)}]^2)}\right\}$$

Hence

$$\begin{aligned} P(W > u) &\approx \textstyle\sum_{i=1}^m \lambda^{(j)} \exp\left\{-\frac{[1-\lambda^{(j)}E(S^{(j)})]u}{\lambda^{(j)}E([S^{(j)}]^2)}\right\} \\ &= \textstyle\sum_{i=1}^m X^{(j)} \exp\left\{-\frac{[1-Z^{(j)}]u}{Y^{(j)}}\right\} \end{aligned} \tag{6}$$

The new optimization problem is obtained by replacing the objective function in (1) by the new objective function given by (6). This is also a non-convex objective function that has properties very similar to the earlier one.

Another setting which one can investigate using the above formulation is the heavy-tailed setting. Assume that the S_i's are heavy-tailed, i.e., $E(e^{\theta S_i}) = \infty$ for all $\theta > 0$. In particular assume that S_i's belong to a particular class of heavy-tailed distributions called sub-exponential distributions (see, e.g., [6] for a precise definition). Most commonly used heavy-tailed distributions, e.g., Pareto, log-normal, Weibull with shape parameter less than 1, belong to this class. Let $F_i(s)$ be the cumulative distribution function (cdf) of S_i. Since all arrrivals are Poisson, the distribution function of the service time faced by server j is given by

$$F^{(j)}(s) = \sum_{i=1}^{n} \lambda_i \alpha_i^{(j)} F_i(s)/\lambda^{(j)}. \tag{7}$$

If F_i's are subexponential then $F^{(j)}(s)$ is also subexponential (infact, $F^{(j)}$ acquires the tail of the heaviest S_i that has $\alpha_i^{(j)} > 0$; see, e.g., [6]) A result for $GI/G/1$ queues with

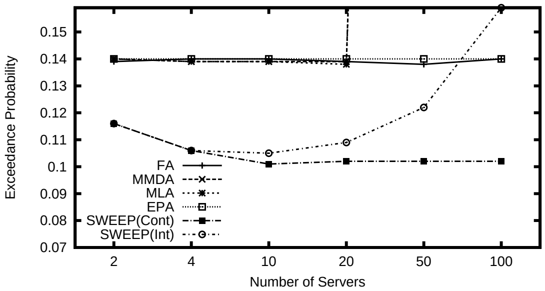


Fig. 7. Exceedance probability for Different Algorithms on Data Set B

FCFS discipline and sub-exponential service time ([19]) says that

$$P(W^{(j)} > u) \sim \frac{\lambda^{(j)} E(S^{(j)})}{(1-\lambda^{(j)} E(S^{(j)})} \frac{1}{E(S^{(j)})} \int_{s=0}^{u} (1 - F^{(j)}(s)) ds \qquad (u \to \infty)$$

In fact, this relationship is valid much more generally; see, e.g., [13] for the case of Markov modulated arrivals. But we restrict ourselves to the $M/G/1$ setting. Using the fact given by (7), the above simplifies to

$$P(W^{(j)} > u) \sim \frac{X^{(j)} Y^{(j)}}{1 - Z^{(j)}} \qquad (u \to \infty).$$

Here $X^{(j)}$ and $Z^{(j)}$ have the same definition as before, but now

$$Y^{(j)} = \sum_{i=1}^{n} \alpha_i^{(j)} \lambda_i \int_{s=0}^{u} (1 - F_i(s)) ds.$$

Hence we have the same optimization problem as in (1), but now the initial data (a_i, b_i, c_i) is defined differently. In particular $(a_i, b_i, c_i) = (\lambda_i, \lambda_i \int_{s=0}^{u} (1 - F_i(s)) ds, \lambda_i E(S_i))$ instead of $(\lambda_i, \lambda_i E(S_i^2), \lambda_i E(S_i),)$.

1) Experimental Results: We also conducted experiments to estimate the exceedance probability for requests of Trace B. Using Q-Q plots we found that the data fitted a lognormal distribution very closely. We estimated the shape and scale parameters of the lognormal distribution for each IP address. We then computed the b_i for each IP address by numerically integrating the cdf of the lognormal distribution. The experiments were then run identically. For lack of space, we report only the balanced load results with Data Set B in Fig. 7. The exceedance probability results establish again the superiority of our algorithm over a suite of baseline techniques. We also note that our algorithm just provides a solution to problems that are expressible in the form given by Eqn. 1 and does not depend on what a_i, b_i, c_i represent. Hence, our algorithm may potentially be used for other assignment problems that can be expressed in this canonical form.

B. Weighted Optimizations for Differentiated QoS

We next consider the case where the web-server farm incurs a penalty for making a request wait, and the penalty per unit wait time is different for different request classes. Let w_i be the penalty per unit of waiting time for class i. Then it is easy to see that in this case, we have

$$E(W) = \sum_{j=1}^{m} \left(\frac{\lambda_w^{(j)}}{\lambda_{total}} \right) \frac{\lambda^{(j)} E([S^{(j)}]^2)}{1 - \lambda^{(j)} E(S^{(j)})} \qquad (8)$$

where $\lambda_w^{(j)} = \sum_{i=1}^{n} \lambda_i \alpha_i^{(j)} w_i$. Hence we get the same optimization problem as in (1), where now the initial data for the problem is $(a_i, b_i, c_i) = (\lambda_i w_i / \lambda_{total}, \lambda_i E(S_i^2), \lambda_i E(S_i))$.

Similarly, in the case of exceedance probabilities for the heavy-tailed case, one can have a different penalty w_i for making a request of class i wait for more than u units of time. Again it is easy to see that we get the same optimization problem as in (1), but with the initial data $(a_i, b_i, c_i) = (\lambda_i w_i / \lambda_{total}, \lambda_i \int_{s=0}^{u} (1 - F_i(s)) ds, \lambda_i E(S_i))$. Since all these problems have the same canonical form that we address, we apply our algorithm in an identical fashion to solve the differentiated QoS setting as well.

VI. Related Work

The stream assignment problem has structural similarities to the popular domains of task assignment and file assignment.

Task Assignment: There exists a significant amount of work on heuristics and optimality proofs for task-level sceduling (see, e.g., [12], [22]) and several references therein), both for the case of known and unknown task durations. In task-level scheduling, each arriving task is dispatched to a server/processor based either on its actual service-time if known, and/or the state of the different queues. Harchol-Balter et. al. [12] (see also [22]) solve the task assignment problem by. apriori partitioning the range of the possible service-times into intervals. Each interval is assigned to one server. Each task is sent to the server assigned to the interval, that its service-time lies in. The intervals are chosen so that the load on each server is equal. In a shared server farm or a cloud, each client is usually assigned to one (or a small number of) servers. Hence, stream assignment becomes a more pertinent problem in this setting.

File Assignment: Lee et. al [15] considered the assignment of files to disks for storage. Disk accesses to each file are assumed to be Poisson with a known rate. The service time for each file access is assumed to be deterministic. The problem is to determine the subset of files to put on each server (without file splitting) so that the load on each server is (roughly) equal, and the overall expected response time for accessing the files is minimized. The solution they propose is again ordering files by their service times, and then partitioning this ordered set so that the load on each server is equal.

Stream Assignment: There has been some work on stream assignment as well ([17], citewhitt). The work in [22] is closest to our setting. It shows through simple examples how combining streams with similar variances (in order to be assigned to the same server) may be beneficial, but does not give specific algorithms for doing that. One may note that file assignment becomes a special case of stream assignment if all requests in a stream have the same service time. Similarly, task assignment is a special case of stream assignment by treating each task as a stream with deterministic service and arrival time. It is also interesting to note that the structure of the solutions to both these problems implicitly partition based on service time (task or file size). One may also note that our solution of partitioning based on variance degenerates to a partitioning based on service time if the service times are fixed. Hence, our algorithm is a generalization of all these techniques for stochastic service times.

In view of the work in [15], our main methodological contribution is going beyond the deterministic service time assumption. It is also worth mentioning here that the extension from deterministic service times to random service times is non-trivial. Infact, the case of deterministic service times just uses the initialization step of the algorithm that we propose. Finally, we are not aware of any work, that formulates and solves models similar to ours in the heavy-tailed setting. An interesting new insight that one gains from this heavy-tailed study, is the following. In the case of finite service-time variance, the (natural) first step of our algorithm is to sort the streams based on the second moments of their service times (or equivalently, just the service times, if they are deterministic, as in [12], [15]). In the case of infinite variance, it turns out that one needs to sort them on the basis of their tail-integrals.

VII. Conclusion

In this work, we have investigated the problem of assiging customer streams to a shared server farm. We precisely solve this problem for the 2-server case and present algorithms for the general case that have a stong theoretical standing. We show by analytic experiments that our algorithms show a significant performance improvement over other policies used in practise and achieve close to optimal performance.

Our work also puts into perspective a lot of work that has gone into fields as diverse as network queue management, task management and file assignment on parallel storage servers. A lot of researchers have proposed various partitioning policies in these fields and shown experimentally that they significantly improve performance. However, to our best knowledge, no theoretical basis has been provided. Whitt [22] partitions customers into service groups and shows that partitioning may help improve performance. Partitioning based on service time is suggested as one of the possiblities. Harchol-balter et al.[12] propose another policy that groups tasks with similar service times together. Lee et al.[15] address the problem of file assignment on parallel storage servers and aim to minimize variance by grouping files with similar access time (same as service time in our scenario). At an algorithmic level, all these approaches are essentially same.

The basic aim of all of these policies has been to reduce variance on a server. However, inadvertently they are doing exactly as the initial partition of our balanced load algorithm. Hence, even though they have not characterized the optimal, they have taken an important first step towards obtaining the optimal solution. Similarly, even though Whitt [22] deals with queues and split based on service times, they inadvertently use the correlation between the second order moment and the first order in the distribution they use. Hence, Our work provides a new insight on these policies and gives them a theoretical standing. Also, we provide a unified framework that covers both task-level and queue-level assignements and characterized the optimal in either cases.

References

[1] Amazon Inc. Amazon Elastic Compute Cloud (EC2) Running IBM. Available at http://aws.amazon.com/ibm/

[2] Brown, L., Gans, N., Mandelbaum, A., Sakov, A., Shen, H., Zeltyn, S., and Zhao, L. (2002). Statistical Analysis of a Telephone Call Center: A Queueing Science Perspective. Technical Report, Wharton School, University of Pennyslvania.

[3] Borst, S., Boxma, O., and Jelenkovic, P.R. (2000): Asymptotic behavior of generalized processor sharing queues with long-tailed traffic sources. In *Proceedings of the INFOCOM*, Tel-Aviv, Israel.

[4] ClarkNet-HTTP trace for one week. Available at ftp://ita.ee.lbl.gov/traces/clarknet_access_log_Sep4.gz

[5] Crovella, M., Taqqu, M.S., and Bestavros, A. (1998): Heavy-tailed probability distributions in the World Wide Web. In *A Practical Guide to Heavy Tails:statistical techniques and applications*, Birkhauser, Boston, 3-25.

[6] Embrechts, P., Kluppelberg, C., and Mikosch, T. (1997): *Modelling Extremal Events*, Springer-Verlag.

[7] EPA-HTT trace for one day. Available at ftp://ita.ee.lbl.gov/traces/epa-http.txt.Z

[8] Fischer, M., Masi, D.M.B, Gross, D., and Shortle, J. (2001): Using quantile estimates in simulating Internet queues with Pareto service times. In *Proc. Winter Simulation Conference*.

[9] Garg, R., Shahabuddin, P., and Verma, A. (2003): On the optimal assignment of streams in server farms. IBM Research Report RI-03-004, Available at http://www.research.ibm.com/resources/paper_search.shtml.

[10] Google Inc. Google App Engine - Google Code. Available at http://code.google.com/appengine/

[11] Gross, D., and Harris, C.M. (1985): *Fundamentals of Queueing Theory,* Second Edition, John Wiley and Sons.

[12] Harchol-Balter, M., Crovella, Mark and Murta, C.D. (1999): On Choosing a Task Assignment Policy for a Distributed Server System. *Journal of Parallel and Distributed Computing* **59**(2), 204-228.

[13] Jelenkovic, P., and Lazar, A.A. (1998): Subexponential asymptotics for a Markov modulated random walk with a queueing application. *Journal of Applied Probability* **35**, 325-347.

[14] Kleinrock, L. (1976): *Queueing Systems, Volume II: Computer Applications.* Wiley, New York.

[15] Lee, L.W., Scheuermann, P. and Vingralek R (2000). File assignment in parallel I/O systems with minimal variance of service time. *IEEE Transactions on Computers* **49**(2), 127-140.

[16] Leland, W.E., Taqqu, M.S., Willinger, W., and Wilson, D.V. (1994): On the self-similar nature of Ethernet traffic. *IEEE/ACM ToN* .

[17] Liu, Z., Squillante, M.S., and Wolf, J.L. (2001): On maximizing service level agreement profits. In *Proc. ACM Electronic Commerce,*.

[18] Naldi, M. (1999): Measurement-based modelling of Internet dial-up access connections. *Computer Networks* **31**, 2381-2390.

[19] Pakes, A.G. (1975): On the tails of waiting-time distributions. *Journal of Applied Probability* **12**, 555-564.

[20] Verma, A., Dasgupta, G., Nayak, T., De., P, and Kothari., R. Server workload analysis for power minimization using consolidation. In *Proc. Usenix ATC*, 2009.

[21] Whitt, W. (1993): Approximations for the GI/G/m queue. *Production and Operations Management* **2**, 114-161.

[22] Whitt, W. (1999): Partitioning customer into service groups. *Management Science* **45** (11), 1579-1592.

Appendix

Lemma 1: The feasible region for this problem is the region R between the two curves $L1$ and $L2$ as shown in Figure 1.

Proof: Consider a point P in the feasible region. Extend the line joining origin to P to meet the line L1 at point Q as shown in Figure 1. Note the Q can be obtained by a convex combination of first k vectors. Also, $P = \beta Q$ where $\beta \in [0, 1]$. So, P can also be obtained by a convex combination of first k vectors and hence is feasible.

Now we show that no point outside the region R can be feasible. The maximum possible value Y^{max} for any feasible solution (X, Y) is given be successively adding vectors (a_1, b_1), (a_2, b_2), $\ldots$, $\beta(a_k, b_k)$ $(0 \leq \beta \leq 1)$ until the sum of their x-components becomes equal to X. To see this, consider any other combination of vectors such that $\sum_{i=1}^{n} \alpha_i a_i = X$ and $\sum_{i=1}^{n} \alpha_i b_i = Y^{max}$. If there is a vector $l > k$ such that $\alpha_l > 0$, then it can be swapped with another vector $h \leq k$, keeping X the same while increasing Y. Similarly, the minimum possible value of Y^{min} is obtained by successively adding the vectors (a_n, b_n), (a_{n-1}, b_{n-1}), $\ldots \beta(a_l, b_l)$ (where $0 \leq \beta \leq 1$), until sum of their x-components becomes equal to X. These points are exactly the same given by curves $L1$ and $L2$. Therefore no feasible solution can lie outside the region R. ∎

Lemma 2: The function $f(x, y) = xy + (A - x)(B - y)$ has convex level-sets in the region $x \leq A/2$, $y \geq B/2$.

Proof: Let $u = A/2 - x$, $v = B/2 - y$. Now, $f(u, v) = AB/2 + 2uv$. To minimize this, uv should be negative. Taking one of the two symmetric solutions, set $u \geq 0$ and $v \leq 0$. This gives $x \leq A/2$ and $y \geq B/2$.

It is sufficient to show that $f(u, v)$ has convex level-sets in the region $u \geq 0$ and $v \leq 0$, which is same as showing that $h(x, y) = -xy$ has convex level-sets in region $x \geq 0$, $y \geq 0$.

Let $x_1 y_1 = x_2 y_2 \geq C$, $x_1, y_1, x_2, y_2 \geq 0$ and $\delta \in [0, 1]$. Now,

$$
\begin{aligned}
C &\leq (\delta + (1 - \delta))^2 x_1 y_1 \\
&= \delta^2 x_1 y_1 + (1 - \delta)^2 x_2 y_2 + \delta(1 - \delta)(x_1 y_2 + x_2 y_1 + (y_1 - y_2)(x_1 - x_2)) \\
&\leq \delta^2 x_1 y_1 + (1 - \delta)^2 x_2 y_2 + \delta(1 - \delta)(x_1 y_2 + x_2 y_1) \\
&= (\delta x_1 + (1 - \delta) x_2)(\delta y_1 + (1 - \delta) y_2).
\end{aligned}
$$

Therefore any convex combination of (x_1, y_1) and (x_2, y_2) is also in the same level set. In case $C \leq x_1 y_1 \leq x_2 y_2$, let $y_2' = x_1 y_1 / x_2 \leq y_2$. Since $x_1 y_1 = x_2 y_2'$, $C \leq (\delta x_1 + (1 - \delta) x_2)(\delta y_1 + (1 - \delta) y_2') \leq (\delta x_1 + (1 - \delta) x_2)(\delta y_1 + (1 - \delta) y_2)$. In this case also the convex combination of (x_1, y_1) and (x_2, y_2) is also in the same level set. ∎

Theorem 1: A feasible solution $\tilde{\alpha}$ is a local minima, if any only if the two servers do not satisfy the exchange property.

Proof: Let $f(\tilde{\alpha}) = X^{(1)} Y^{(1)} + (A - X^{(1)})(B - Y^{(1)})$ where $X^{(1)}$ and $Y^{(1)}$ are given in terms of $\tilde{\alpha}$ as follows:

$$
X^{(1)} = \sum_{i=1}^{n} \tilde{a}_i \tilde{\alpha}_i, \qquad X^{(2)} = A - X^{(1)} \tag{9}
$$

$$
Y^{(1)} = \sum_{i=1}^{n} \tilde{b}_i \tilde{\alpha}_i, \qquad Y^{(2)} = B - Y^{(1)} \tag{10}
$$

The function $f()$ represents the objective function value in terms of the assignment variables $\tilde{\alpha}$.

We first prove the converse part of the theorem. Let $\tilde{\alpha}$ be a local minima. We show that the two servers do not satisfy the exchange property. Assume for contradiction that servers satisfy the exchange property using streams i and i'. Construct a new feasible solution α' from $\tilde{\alpha}$ as follows:

$$
\begin{aligned}
\alpha_i'^{(1)} &= \tilde{\alpha}_i^{(1)} - \delta \qquad & \alpha_{i'}'^{(1)} &= \tilde{\alpha}_{i'}^{(1)} + \delta \\
\alpha_{i'}'^{(2)} &= \tilde{\alpha}_{i'}^{(2)} + \delta \qquad & \alpha_i'^{(2)} &= \tilde{\alpha}_i^{(2)} - \delta.
\end{aligned}
$$

Change in objective function value while moving from $\tilde{\alpha}$ to α' is given by:

$$
\begin{aligned}
f(\alpha') - f(\tilde{\alpha}) &= \delta[(a_{i'} - a_i)(Y^{(1)} - Y^{(2)}) + (b_{i'} - b_i)(X^{(1)} - X^{(2)})] \\
&\quad + 2\delta^2 (a_{i'} - a_i)(b_{i'} - b_i).
\end{aligned}
$$

Since the servers satisfy the exchange property with respect to streams i and i', the solution α' is feasible for sufficiently small values of δ. Also, the change in objective function value is negative for sufficiently and arbitrarily small values of δ. This means that $\tilde{\alpha}$ is not the local minima, contradicting our assumption. Therefore, the two servers do not satisfy the exchange property.

We now prove the first part of the theorem. Let w be a $2n$ dimensional vector specifying a direction of movement in the solution space $\tilde{\alpha}$. Define gradient of $f()$ in direction w as:

$$
g(\tilde{\alpha}, w) = \sum_{i=1}^{n} \sum_{j=1}^{2} \frac{\partial f(\tilde{\alpha})}{\partial \tilde{\alpha}_i^{(j)}} w_i^{(j)}
$$

Consider a feasible solution $\tilde{\alpha}$ such that the two servers do not satisfy the exchange property. Assume for contradiction that $\tilde{\alpha}$ is not a local minima. Therefore there is a vector w such that $\tilde{\alpha} + \delta w$ is a feasible solution for sufficiently and arbitrarily small values of δ and $g(\tilde{\alpha}, w) < 0$. For the feasibility of $\tilde{\alpha} + \delta w$ the following conditions must be satisfied:

$$
\forall i : w_i^{(1)} + w_i^{(2)} = 0 \tag{11}
$$

$$
\forall j : \sum_{i=1}^{n} w_i^{(j)} = 0. \tag{12}
$$

Define the basis vector $e(ik)$ as a vector with all the components zero except the following:

$$
\begin{aligned}
e(ik)_i^{(1)} &= -1 \quad & e(ik)_i^{(2)} &= 1 \\
e(ik)_k^{(1)} &= 1 \quad & e(ik)_k^{(2)} &= -1.
\end{aligned}
$$

Note that the basis vector $e(ik)$ represents the exchange of streams i and k between the two servers. We first decompose w using these basis vectors and then show that there is a pair of stream that satisfies the exchange property.

Lemma 3: The vector w can be decomposed into basis vectors as $w = \sum_{i,k} w_{ik} e(ik)$ such that:

$$
w_{ik} \geq 0 \text{ and } w_{ik} > 0 \Rightarrow w_i^{(1)} < 0, w_k^{(2)} < 0. \tag{13}
$$

Proof: Consider i such that $w_i^{(1)} < 0$. Using (11) we have $w_i^{(2)} > 0$. This, along with (12) implies that there exists k such that $w_k^{(2)} < 0$. Set $w_{ik} = \min(-w_i^{(1)}, -w_k^{(2)})$. Update $w = w - w_{ik}e(ik)$. Note that by such an update either $w_i^{(1)}$ or $w_k^{(2)}$ becomes zero. Also note that the signs of $w_i^{(1)}$ and $w_k^{(2)}$ are unchanged. Repeating this, gives the desired decomposition. ∎

Lemma 4: Consider a decomposition of w into basis vectors satisfying (13). If $\tilde{\alpha} + \delta w$ is feasible then so is $\tilde{\alpha} + \delta w_{ik} e(ik)$.

Proof: The above lemma is trivially true of $w_{ik} = 0$. Note that the solution $\tilde{\alpha} + \delta w_{ik} e(ik)$ does not violate the balanced load constraint (5). We only need to show that

$$\tilde{\alpha}_i^{(1)} - \delta w_{ik} \geq 0 \text{ and } \tilde{\alpha}_k^{(2)} - \delta w_{ik} \geq 0$$

Since w_{ik} is a decomposition of w into the basis vectors satisfying (13), if $w_{ik} > 0$ then $w_i^{(1)} < 0$ and $w_k^{(2)} < 0$. Since $\tilde{\alpha} + \delta w$ is feasible:

$$\tilde{\alpha}_i^{(1)} + \delta w_i^{(1)} \geq 0$$
$$\Rightarrow \tilde{\alpha}_i^{(1)} - \delta \sum_{l=1}^{n} w_{il} \geq 0$$

Since, $w_{il} \geq 0$ for all l, we have:

$$\tilde{\alpha}_i^{(1)} - \delta w_{ik} \geq 0.$$

Using a symmetric argument it can also be shown that $\tilde{\alpha}_k^{(2)} - \delta w_{ik} \geq 0$. ∎

Since $g(\tilde{\alpha}, w) < 0$, and $w = \sum_{i,k} w_{ik} e(ik)$ we have:

$$\sum_{i=1}^{n} \sum_{k=1}^{n} w_{ik} g\left(\tilde{\alpha}, e(ik)\right) < 0$$
$$\Rightarrow \quad \exists i, k : w_{ik} g\left(\tilde{\alpha}, e(ik)\right) < 0$$

Substituting the value of function $g()$, vectors $e(ik)$, and the fact that:

$$\frac{\partial f(\tilde{\alpha})}{\partial \tilde{\alpha}_i^{(j)}} = \tilde{a}_i Y^{(j)} + \tilde{b}_i X^{(j)},$$

we get:

$$(-\tilde{a}_i)Y^{(1)} + a_k Y^{(1)} - b_i X^{(1)} + b_k X^{(1)} + a_i Y^{(2)} - a_k Y^{(1)} + b_i X^{(2)} - b_k X^{(1)} < 0$$

Simplifying the above expression gives:

$$(\tilde{a}_k - \tilde{a}_i)(Y^{(j)} - Y^{(l)}) + (\tilde{b}_k - \tilde{b}_i)(X^{(j)} - X^{(l)}) < 0.$$

Since $w_{ik} > 0$ and $\tilde{\alpha} + \delta w_{ik} e(ik)$ is a feasible solution, $\tilde{\alpha}_i^{(1)} > 0$ and $\tilde{\alpha}_k^{(2)} > 0$. This means that the two servers satisfy the exchange property. This is a contradiction to our assumption that $\tilde{\alpha}$ is not a local minima. Therefore, $\tilde{\alpha}$ must be a local minima. This completes the proof. ∎

Theorem 2: If the current partition is same as the target partition then the solution is optimal.

Proof: Assume for contradiction that the solution is not optimal. Therefore, there are two streams i and l that satisfy the exchange property i.e. $\tilde{\alpha}_i > 0$ and $\tilde{\alpha}_l < c_l$ and:

$$(\tilde{a}_i - \tilde{a}_l)(Y^{(2)} - Y^{(1)}) + (\tilde{b}_i - \tilde{b}_l)(X^{(2)} - X^{(1)}) < 0.$$

This gives $\tilde{b}_i - \tilde{b}_l < \gamma(\tilde{a}_i - \tilde{a}_l)$. Since $\tilde{\alpha}_i > 0$ and the target partition is same as the current partition, i lies above the line defining the partition. Therefore, $\tilde{b}_i - \tilde{b}_k \geq \gamma(\tilde{a}_i - \tilde{a}_k)$. Similarly, since $\tilde{\alpha}_l < c_l$, l lies below the line i.e. $\tilde{b}_l - \tilde{b}_k \leq \gamma(\tilde{a}_l - \tilde{a}_k)$. This gives, $\tilde{b}_i - \tilde{b}_l \geq \gamma(\tilde{a}_i - \tilde{a}_l)$ which is a contradiction to our assumption. ∎

Theorem 3: If current partition is not the same as the target partition, then the solution is not optimal.

Proof: Assume that stream k is shared between the two servers in the partitions. If current partition is not same as the target partition, then either (a) there is stream i (not equal to k) that is assigned entirely to the first server in the target partition and to the second server in the current partition, or (b) there is a stream i assigned entirely to the second server in the target partition and first server in the current partition. Without loss of generality, we consider the former case. We now show that streams i and k satisfy the exchange property in the current partition.

Since i belongs to the second server and k is shared in the current partition, $\tilde{\alpha}_i < c_i$ and $\tilde{\alpha}_k > 0$. Since i belongs completely to the first server in the target partition, it must be strictly above the line defining the the target partition, i.e. $\tilde{b}_i - \tilde{b}_k > \gamma(\tilde{a}_i - \tilde{a}_k)$, which is same as the exchange property.

Since the streams satisfy the exchange property, the current solution cannot be the optimal. ∎

Theorem 4: The algorithm terminates in $O(n^2 \log n)$ steps.

Proof: Initially, (from our assumption) the slope of the current partition is larger than that of the target partition. The two partitions become identical when their slopes become the same. It can be verified that after each exchange the slope of the target partition increases while the slope of the current partition decreases. Each exchange is defined by exactly a pair of two streams in the plane. There are at most $n(n-1)/2$ such pairs. Therefore there are at most $O(n^2)$ exchanges.

For each exchange, we need to find a stream i that maximizes the slope $(\tilde{b}_i - \tilde{b}_k)/(\tilde{a}_i - \tilde{a}_k)$. This can be done in $O(\log n)$ time by maintaining a sorted list of $n-1$ points for every stream. The list corresponding to stream k is sorted by $(\tilde{b}_i - \tilde{b}_k)/(\tilde{a}_i - \tilde{a}_k)$. This gives the desired bound. ∎

Lemma 5: A solution is feasible iff all the corresponding $m-1$ points lie inside the region R.

Proof: Consider a solution where kth point lies outside the region R. Let $X = \sum_{j=1}^{k} X^{(j)}$. Using an argument similar to that in proof of Lemma 1 it can be shown that any other feasible solution with $\sum_{j=1}^{k} X'^{(j)} = X$, should have $Y^{min} \leq \sum_{j=1}^{k} Y'^{(j)} \leq Y^{max}$ where Y^{max} (and Y^{min}) is obtained by accumulating streams in decreasing (increasing) order of slopes until the total x-component becomes equal to X. Therefore, this cannot be a feasible solution.

Now, consider a solution consisting of a sequence of $m-1$ points in decreasing order of their slopes as shown in Figure 4. Using induction we show that it is feasible. Consider the first point $P1$ in the solution. Join O to $P1$ and extend it to meet the boundary of the region R at point $Q1$. Now, remove the first server and the streams assigned to it (which is a fraction of the streams that form $Q1$) to get another problem of size $m-1$. The origin of new region shifts to point $P1$. The boundary of new region remains unchanged after $Q1$. The boundary between $P1$ and $Q1$ becomes inflated as shown in the figure. Since the slope of $P1-P2$ (and all subsequent pairs after that) is less than that of $O-P1$, all the remaining points still remain inside the new feasible region. From induction hypothesis, that the remaining $m-2$ points represent a feasible solution in the new problem. Therefore the $m-1$ points represent a feasible solution in the original problem. ■

Lemma 6: The optimal solution will have all the $m-1$ points on the boundary of the region R.

Proof: Consider a feasible solution for m servers sorted according to slopes (Y/X) represented using $m-1$ points as discussed earlier. Join the k^{th} point to the $k+1^{st}$ point using the vectors of streams present on the $k+1^{st}$ server in decreasing order of their slopes. It suffices to show that the curve so obtained is convex (since the convex curve obtained by adding all the stream vectors is unique and is precisely the boundary of feasible region).

We prove this using induction on number of servers. The base case of two servers has already been shown to be true. Now, the curve obtained from the first $m-2$ points is convex using the induction hypothesis. Similarly the curve obtained from the last $m-2$ points is also convex. Therefore, the curve obtained from all the $m-1$ points is convex. Therefore, it has to be identical to the boundary of feasible region. Therefore, points corresponding to the optimal solution lie on the boundary of the feasible region. ■