Task Assignment in a Distributed System: Improving Performance by Unbalancing Load

Mark E. Crovella[†]
Department of Computer Science
Boston University
Boston, MA 02215
crovella@bu.edu

Mor Harchol-Balter[*]
Laboratory for Computer Science
MIT, NE43-340
Cambridge, MA 02139
harchol@theory.lcs.mit.edu

Cristina D. Murta[‡]
Department of Computer Science
Boston University
Boston, MA 02215
murta@bu.edu

October 31, 1997

BUCS-TR-1997-018

Abstract

We consider the problem of task assignment in a distributed system (such as a distributed Web server) in which task sizes are drawn from a heavy-tailed distribution. Many task assignment algorithms are based on the heuristic that balancing the load at the server hosts will result in optimal performance. We show this conventional wisdom is less true when the task size distribution is heavy-tailed (as is the case for Web file sizes). We introduce a new task assignment policy, called Size Interval Task Assignment with Variable Load (SITA-V). SITA-V purposely operates the server hosts at different loads, and directs smaller tasks to the lighter-loaded hosts. The result is that SITA-V provably decreases the mean task slowdown by significant factors (up to 1000 or more) where the more heavy-tailed the workload, the greater the improvement factor. We evaluate the tradeoff between improvement in slowdown and increase in waiting time in a system using SITA-V, and show conditions under which SITA-V represents a particularly appealing policy. We conclude with a discussion of the use of SITA-V in a distributed Web server, and show that it is attractive because it has a simple implementation which requires no communication from the server hosts back to the task router.

[*]Supported by the NSF Postdoctoral Fellowship in the Mathematical Sciences.
[†]Supported in part by NSF Grants CCR-9501822 and CCR-9706685.
[‡]Supported by a grant from CAPES, Brazil. Permanent address: Depto. de Informática, Universidade Federal do Paraná, Curitiba, PR 81531, Brazil. Email: cristina@dcc.ufmg.br.

1 Introduction

Increasingly, high performance servers are being implemented as distributed systems – collections of loosely-coupled hosts, as shown in Figure 1. In such a system, each arriving task must be assigned to some particular host for service, and the policy used to make that assignment can have a significant impact on performance. Many task assignment policies have the goal of balancing the load across the hosts of the system. This heuristic is reasonable because balancing load often minimizes the expected waiting time for tasks in the system.

In contrast, in this paper we explore systems for which balancing load is *not* necessarily the best policy. In particular, we are concerned with a distributed server which uses the processor-sharing (PS) discipline at the hosts. An important aspect of the systems we consider is that task sizes are drawn from a *heavy-tailed* distribution. A heavy-tailed distribution is one whose tail declines like a power-law, that is, $P[X > x] \sim x^{-\alpha}$ for $0 < \alpha \leq 2$. Heavy-tailed distributions are increasingly being observed in a wide range of computer workloads, see Section 7. Since task sizes in our case follow heavy-tailed distributions, their service demands show extremely high variability. Most importantly, heavy-tailed task size distributions have the property that while the overwhelming majority of tasks are very small, more than half of the load is made up by a tiny minority of the the very largest tasks. Another important aspect of the systems we consider is that it is possible to determine a task's service demand upon its arrival to the system. Our motivating application is the design of a distributed Web server—since first, Web file sizes appear to exhibit heavy tailed distributions [6, 2] and second, task sizes may in most cases be inferred from the names of the files being requested.

The metrics by which we judge system performance are user-oriented metrics: mean waiting time and mean slowdown, where slowdown is the ratio of a task's waiting time to its service demand. System designers have often focused on developing policies that minimize mean waiting time. However, in this paper we focus on minimizing slowdown, because slowdown translates more directly to user-perceived performance. Minimizing waiting time tends to improve system performance for large tasks (since these contribute most to waiting time) but may not have much effect on short tasks. As a result, in a system with highly variable task sizes, minimizing mean waiting time alone can nonetheless cause users to wait for extremely long periods for short tasks to complete. In contrast, minimizing slowdown tends to improve the performance of all jobs equally in proportion to their demand.

In this paper we show that when task sizes are heavy-tailed, policies that balance load in fact do a poor job of minimizing slowdown. We demonstrate this fact by introducing a new task assignment policy for distributed servers called **SITA-V** (Size Interval Task Assignment with Variable Load) which operates different hosts at different loads. A server using SITA-V inspects each incoming task as it arrives, and assigns it to some host based on the task's size. SITA-V exploits the heavy-tailed task size distribution by running the overwhelming majority of tasks (all small-sized tasks) on hosts which are loaded below the average system load and running the tiny minority (all large-sized tasks) on a host which is loaded above the average system load. We show that using this strategy, SITA-V can reduce mean task slowdown to levels far below that of traditional, load-balancing policies.

In improving mean slowdown, observe that SITA-V does not sacrifice throughput. The overall system utilization (ρ) is not changed; load is merely shifted among hosts. However, SITA-V achieves

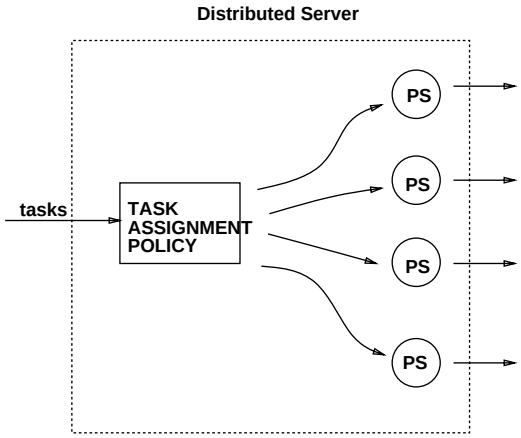


Figure 1: Model for distributed server with task assignment policy (a "server farm").

its result at some cost: an increase in mean task waiting time. We evaluate the tradeoff between reducing slowdown and increasing waiting time in a system employing SITA-V. We show that the two factors that determine the nature of the tradeoff are the variability of tasks (as measured by the exponent α in the task size distribution) and the overall system utilization ρ. In general, we find that when task sizes are highly variable ($\alpha < 1$) SITA-V can result in remarkable improvements in slowdown — by factors as great as 1000 or more. However we also show that under some conditions, the improvements under SITA-V can come at a high cost in increased waiting time. We conclude that SITA-V is a particularly appealing policy when α is small; in that case SITA-V results in significant improvements in slowdown while imposing relatively smaller costs in terms of additional waiting time.

Finally, we return to our motivating example and discuss applicability of SITA-V in the context of modern distributed Web servers. We show that modern distributed Web servers all assume that load should be balanced among the server hosts; and we discuss the potential benefits presented by the deployment of SITA-V in such systems.

The remainder of this paper is structured as follows. In Section 2 we describe the system model that we use. In Section 3 we present analysis of the balanced-load case, to form a comparison for what follows. Then in Section 4 we describe the SITA-V policy, and explain how it achieves minimal slowdown, and in Section 5 we evaluate the tradeoff between slowdown and waiting time under SITA-V. Next in Section 6 we discuss applicability of SITA-V in distributed Web servers. Finally in Section 7 we discuss related work, and in Section 8 we present our conclusions.

2 Distributed Server Model

Our model of a distributed server assumes h identical hosts. Tasks arrive to the system according to a Poisson process[2] with rate λ. A task corresponds to a request for service. A *task-assignment*

[2]We acknowledge that assuming a Poisson arrival process is unrealistic — for web servers the arrival process is usually more bursty than a Poisson Process.

policy is an algorithm which assigns each task to one of the hosts for service.

Heavy-Tailed distributions. One of the motivating factors in this work is the recent observation that file sizes on Web servers often exhibit *heavy-tailed* distributions [2, 7, 6]. We say here that a random variable X follows a heavy-tailed distribution (with tail index α) if

$$P[X > x] \sim x^{-\alpha}, \quad \text{as } x \to \infty, \;\; 0 < \alpha < 2,$$

where $a \sim b$ means that $\lim_{x\to\infty} a/b = c$ for some constant c. The simplest heavy-tailed distribution is the *Pareto* distribution, with probability mass function

$$f(x) = \alpha k^{\alpha} x^{-\alpha-1}, \quad \alpha, k > 0, \;\; x \geq k,$$

and cumulative distribution function

$$F(x) = P[X \leq x] = 1 - (k/x)^{\alpha}.$$

Heavy-tailed distributions are characterized by extremely high variability, which increases sharply as α decreases. Such a distribution has infinite variance; and if $\alpha \leq 1$ the distribution has infinite mean.

Recent work has shown that Web file sizes on servers often show heavy-tailed size distributions. Typical values of the tail index α seem to be in range of 1.1 to 1.3 [7, 6]. These values of α are so low as to motivate particular focus on analyzing the effects of high variability in file size, and on developing resource management policies that specifically address high variability in file size, which we do in this paper.

Task sizes. We assume that task sizes show some maximum (but large) value. Note that this would be the expected case for a Web server, which would have some largest file. As a result, we model task sizes using a distribution that follows a power law, but has an upper bound. We refer to this distribution as a *Bounded Pareto.* It is characterized by three parameters: α, the exponent of the power law; k, the smallest possible observation; and p, the largest possible observation. The probability mass function for the Bounded Pareto $B(k, p, \alpha)$ is defined as:

$$f(x) = \frac{\alpha k^{\alpha}}{1 - (k/p)^{\alpha}} \, x^{-\alpha-1} \quad k \leq x \leq p. \tag{1}$$

The shape of $f(x)$ is shown in Figure 2 (right). If X is a random variable drawn from a $B(k, p, \alpha)$ distribution, then the jth moment of X (for $\alpha \neq j$) is given by:

$$\mathbf{E}\left\{X^j\right\} = \frac{\alpha k^{\alpha} \left(k^{j-\alpha} - p^{j-\alpha}\right)}{(\alpha - j)\left(1 - (k/p)^{\alpha}\right)} \tag{2}$$

Note that the Bounded Pareto distribution has all its moments finite. Thus, it is not a heavy-tailed distribution in the sense we have defined above. However, this distribution will still show high variability if $k \ll p$. For example, Figure 2 (left) shows the second moment $\mathbf{E}\{X^2\}$ of this

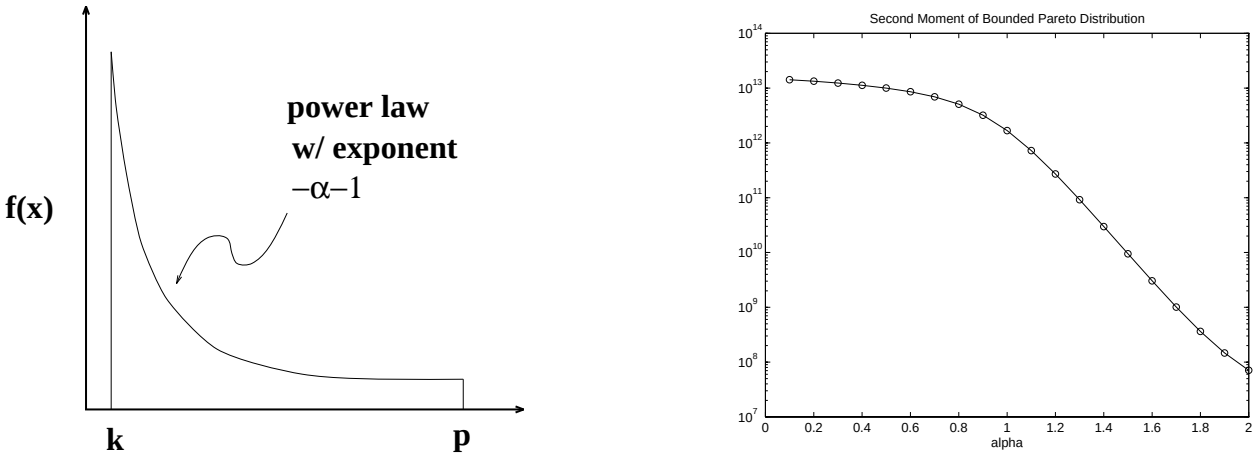


Figure 2: Parameters of the Bounded Pareto Distribution (left); Second Moment of $B(k, 10^{10}, \alpha)$ as a function of α, when $\mathbf{E}\{X\} = 3000$ (right).

distribution as a function of α for $p = 10^{10}$, where k is chosen to keep $\mathbf{E}\{X\}$ constant at 3000 ($0 < k \leq 1500$). The figure shows that the second moment explodes exponentially as α declines.

Finally, we make the additional simplifying assumption that the service times of the tasks are independent; in particular, the sizes of the ith and $i+1$st arriving tasks are uncorrelated.

Known task sizes. We assume that a centralized task router is able to inspect the service demands of each arriving task. The fact that in this model of a distributed server a task's service time is known at the time of its arrival greatly differentiates this model from a typical network of workstations load balancing model where task sizes are *not* known in advance and must be estimated as the task runs (*e.g.,* see [13]). The design of a centralized router for distributed Web servers that can inspect task service demands is described in [14].

Performance metrics. The effectiveness of the SITA-V task assignment scheme will be measured in terms of mean waiting time, $\mathbf{E}\{W\}$, and mean slowdown, $\mathbf{E}\{S\}$, as defined below:

Waiting time (W). The waiting time for a task is defined as the time from when the task arrives at the system until the task leaves the system, minus its service time. We evaluate the *mean waiting time*, i.e., the average task waiting time, over all tasks. A related commonly used metric is *mean response time* which is the sum of mean waiting time plus mean service time.

Slowdown (S). A task's slowdown is its waiting time divided by its size (execution time). That is, the task's waiting time is normalized by its size. Minimizing slowdown is often cited as the most important performance objective because it results in short tasks waiting less time and long tasks waiting longer.

Note that if task sizes are denoted by X, then mean slowdown is:

$$\mathbf{E}\{S\} = \mathbf{E}\{W/X\},$$

where W and X are not independent for the PS discipline.

If we consider tasks to be analagous to files being served, then task size can be thought of in units of bytes. Then we can also define a metric B, per-byte slowdown, where the per-byte slowdown of a task is the task's slowdown multiplied by its file size. Observe that

$$\mathbf{E}\{B\} = \mathbf{E}\{SX\} = \mathbf{E}\{WX/X\} = \mathbf{E}\{W\},$$

where all expectations are task averages.

This means that minimizing waiting time can be thought of as minimizing *per-byte* slowdown; and that minimizing (what we are calling) slowdown can be thought of as minimizing *per-file* slowdown.

Of these two metrics (slowdown and waiting time), we consider slowdown to be most important, because it is desirable that a task's delay be proportional to its size. That is, in a system in which task sizes are highly variable, users are likely to anticipate short delays for short tasks, and are likely to tolerate long delays for longer tasks. This may be especially true in the case of Web requests, where users may typically have a feeling for the rough order of magnitude of their request.

Parameteric ranges used in evaluation. In evaluating SITA-V throughout the paper we choose specific values or ranges for the system parameters, as summarized in Table 1. These values are chosen with the aim of being realistic for a medium-scale distributed system with heavy-tailed workload, such as a distributed Web server.

Number of hosts	$h = 2 - 16$
System load	$\rho = .2 - .9$
Mean service time	$\mathbf{E}\{X\} = 3000$ time units
Task arrival rate	$\lambda = \rho \cdot \frac{1}{\mathbf{E}\{X\}} \cdot h$ tasks/unit time
Maximum task service time	$p = 10^{10}$ time units
α parameter	$.5 < \alpha \leq 2$
Minimum task service time	chosen so that mean task service time stays constant as α varies ($0 < k \leq 1500$)

Table 1: Parameters used in evaluating task assignment policies

We consider a range in the number of hosts h from 2 to 16, representative of a medium-scale distributed system and a range in the overall system load ρ from lightly loaded to heavily loaded. The mean task execution time, $\mathbf{E}\{X\}$, is chosen to be 3000 time units. This could be interpreted as the number of bytes in an average Web page [8]. The arrival rate λ is then determined by the above formula.

For completeness we consider a range of α from .5 to 2; as discussed in Section 2 this means that task variability spans a wide range from moderately variable ($\alpha = 2$) to highly variable ($\alpha \approx 0$). Recent measurements indicate that a Web server might experience α in the range of approximately 1.1 to 1.3 [7].

If k and p are held constant, an increase in α results in a decrease in the mean task size. Therefore, to hold $\mathbf{E}\{X\}$ constant, we decrease k, the lower bound on the $B(k, p, \alpha)$ distribution,

as we increase α. In practice, changing p has a relatively less pronounced effect on $\mathbf{E}\{X\}$, so we choose to hold p constant at 10^{10}. Again, this value corresponds to a reasonable value for a largest file on a Web server, measured in bytes. Thus α and k are throughout defined in pairs, where

$$3000 = \mathbf{E}\{X\} = \frac{\alpha k^{\alpha}(k^{1-\alpha} - p^{1-\alpha})}{(\alpha - 1)(1 - (k/p)^{\alpha})}. \tag{3}$$

As α ranges from .5 to 2, k ranges from just slightly greater than 0 (time units) to 1500 (time units).

3 Performance of Distributed Server With Balanced Load

In this section we review the mean slowdown for a distributed server with any task assignment policy which balances the expected load at the server hosts. For the purposes of analysis, we assume that the arrival process at each host is Poisson.

We begin by reviewing the analysis of the M/G/1 PS queue. Recall that in a M/G/1 PS queue all tasks experience the same slowdown, independent of their size. Specifically, (see [17]),

$$\mathbf{E}\{W|tasksize = x\} = \frac{\rho}{1-\rho} \cdot x, \tag{4}$$

where W denotes the waiting time for the M/G/1 PS queue and ρ denotes the expected load. Equation 4 can be used to derive $\mathbf{E}\{W\}$ and $\mathbf{E}\{S\}$ for the M/G/1 PS:

$$\begin{aligned} \mathbf{E}\{W\} &= \frac{\rho}{1-\rho}\mathbf{E}\{Tasksize\} \\ \mathbf{E}\{S\} &= \frac{\rho}{1-\rho} \end{aligned}$$

Now consider a distributed server employing any task assignment policy which balances the expected load among its hosts. Then if ρ denotes the system load, then the load at every host is also ρ, so the mean slowdown at each host is $\rho/(1-\rho)$ by the above analysis. Thus the mean slowdown for the system is $\rho/(1-\rho)$. This fact holds independently of the distribution of task sizes, and also independently of the particular task assignment policy.

4 Improving Performance by Unbalancing Load: SITA-V

In the previous section we studied a distributed system of h PS server hosts employing any task assignment policy which balances the load at the server hosts. We found that this distributed system has the same mean slowdown as a single PS server of equal power to the h hosts combined.

In this section we ask whether it is possible to improve upon the performance of such a distributed server system. The answer is yes—by unbalancing the load at the server hosts.

We introduce a task assignment policy called SITA-V: Size Interval Task Assignment with Variable Load. SITA-V is based on the following idea: mean slowdown could be reduced if many tasks were run on a host with reduced load (and therefore low mean slowdown), while the remaining few tasks were run on a host with high load. For traditional task size distributions this doesn't seem possible because the host with many tasks also will experience high load and the host with few tasks will experience low load. However, when the task size distribution is heavy-tailed the *reverse* is often true. A tiny fraction of the very largest tasks can constitute more than half the load—hence the name "heavy-tail". The more heavy-tailed the task size distribution (*i.e.,* the smaller α) the smaller is this tiny fraction, and thus the greater the number of tasks which are benefitted by running at a reduced load.

To see concretely exactly how the algorithm operates, we present in the next section the SITA-V algorithm for the case of just 2 hosts: SITA-V_2. Then in Section 4.2 we present the general SITA-V algorithm.

4.1 SITA-V in the 2-Host Case

SITA-V_2 is defined as follows. Let k denote the smallest possible task size, let p denote the largest possible task size, and let x_1 be some number between k and p. Under SITA-V_2 all tasks of size between k and x_1 are assigned to host 1, all tasks of size between x_1 and p are directed to host 2, and the cutoff x_1 is uniquely defined so as to minimize $\mathbf{E}\{S\}$ for the system. Note that legal values of x_1 must respect the constraint that the load on both hosts must stay below 1.

An important special case of SITA-V_2 is EQ-LOAD where x_1 is chosen so that the load on the two hosts is equal, $x_1 = x_e$. We will use EQ-LOAD as a comparison case for SITA-V_2 because it is representative of load-balancing schemes. We begin by presenting a thought-experiment to explain why it is that SITA-V_2 works and what precisely it is doing. Then we will present a collection of graphs which demonstrate this intuition in action for a range of αs and ρs.

Consider a 2-host system in which tasks of size less than x_1 are sent to host 1, and tasks of size greater than x_1 are sent to host 2. Let p_i $(i = 1, 2)$ be the fraction of tasks that are assigned to host i, and let S_i be the slowdown for tasks assigned to host i. Then:

$$\mathbf{E}\{S\} = p_1\mathbf{E}\{S_1\} + p_2\mathbf{E}\{S_2\} \tag{5}$$

where the values of p_i and $\mathbf{E}\{S_i\}$ depend on the cutoff point x_1. Then, when $x_1 = x_e$, we have $\mathbf{E}\{S_1\} = \mathbf{E}\{S_2\}$ and $\mathbf{E}\{S\} = \mathbf{E}\left\{S^{EQ-LOAD}\right\}$. Figure 3 depicts $\mathbf{E}\{S_1\}$ and $\mathbf{E}\{S_2\}$ as a function of the cutoff, x_1. $\mathbf{E}\{S_1\}$ increases as the cutoff x_1 is increased, and $\mathbf{E}\{S_2\}$ decreases as x_1 is increased.

First we ask the question: might $\mathbf{E}\{S\}$ be lower for some value of $x_1 \neq x_e$? To answer this, suppose that when $x_1 = x_e$, p_1 is very large—*i.e.,* $p_1 \approx 1$. Then if we decrease x_1 by a small amount, it will still be the case that $p_1 \approx 1$ and $p_2 \approx 0$. In that case $\mathbf{E}\{S\}$ will be close to the $\mathbf{E}\{S_1\}$ curve, because $\mathbf{E}\{S\}$ will be dominated by $\mathbf{E}\{S_1\}$ (as shown by Equation 5). But that means the value of $\mathbf{E}\{S\}$ will be below the value at the crosspoint. That is, the system will have achieved a lower value of $\mathbf{E}\{S\}$.

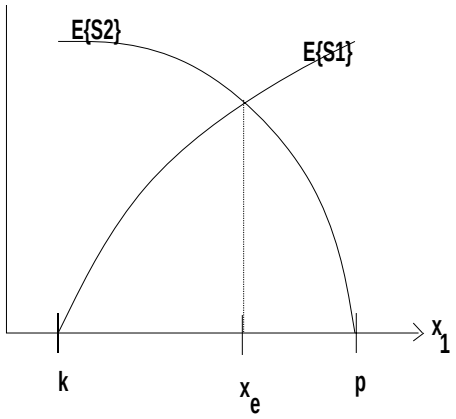


Figure 3: $\mathbf{E}\{S_1\}$ and $\mathbf{E}\{S_2\}$ shown as a function of the cutoff x_1. The point $x_1 = x_e$ depicts balanced load among the hosts (EQ-LOAD).

Now we examine the question of whether it is in fact the case that $p_1 \approx 1$ when $x_1 = x_e$. Remember that half of the work is contained in the tasks whose size is larger than x_e. Therefore p_1 will be close to 1 only when those largest tasks (containing half the work) constitute a tiny fraction of all tasks. That is, $p_1 \approx 1$ if half the work is contained in only a tiny fraction of the largest tasks—which in fact is exactly the case for heavy-tailed distributions.

The fraction of largest tasks containing half the work decreases as α decreases—consistent with the idea that tails grow "heavier" with decreasing α. So, the lower the α parameter, the greater is p_1 when $x_1 = x_e$. For example, when $\alpha = 1.1$, $p_1 = .997$, while when $\alpha = .5$, p_1 increases to .9999996. This is reflected in the relative improvement of SITA-V_2 over EQ-LOAD: for $\alpha = 1.1$, the improvement is about a factor of 2, while for $\alpha = 0.5$, the improvement is about a factor of 33.

Figure 4 shows SITA-V_2's improvement over EQ-LOAD using the same type of drawing as Figure 3 over a range of αs and a range of ρs, so that we can better understand the effect of α and ρ on SITA-V_2's performance. Each graph in the figure plots $\mathbf{E}\{S_1\}, \mathbf{E}\{S_2\}$, and $\mathbf{E}\{S\}$; different graphs in the figure are for different values of α and ρ. In each row of graphs, system load (ρ) is held constant; the top row has $\rho = 0.3$, the next rows have $\rho = 0.5$ and 0.7, and the lowest row has $\rho = 0.9$. In each column, task size variability (measured by α) is held constant; α varies from 0.7 on the left, through 1.1 and 1.3, to 1.9 on the right.

In each graph, the dotted line moving from bottom left to top right represents $\mathbf{E}\{S_1\}$. $\mathbf{E}\{S_1\}$ increases as x_1 increases because more load is then sent to host 1. The dotted line moving from top left to bottom right represents $\mathbf{E}\{S_2\}$, which decreases as x_1 increases because then less load is directed to host 2.

Note from Figure 4 that the crosspoint (x_e) depends only on α. As explained above, the smaller the α parameter, the further right is the crosspoint. Now the important point is that since the crosspoint is larger for smaller α, p_1 is also larger. Therefore there is more room for improvement; *i.e.,* $\mathbf{E}\{S\}$ is dominated by $\mathbf{E}\{S_1\}$ over a larger region, when moving left from x_e, as illustrated in the graphs.

Now consider the effect of increasing ρ. SITA-V_2's improvement factor with respect to mean

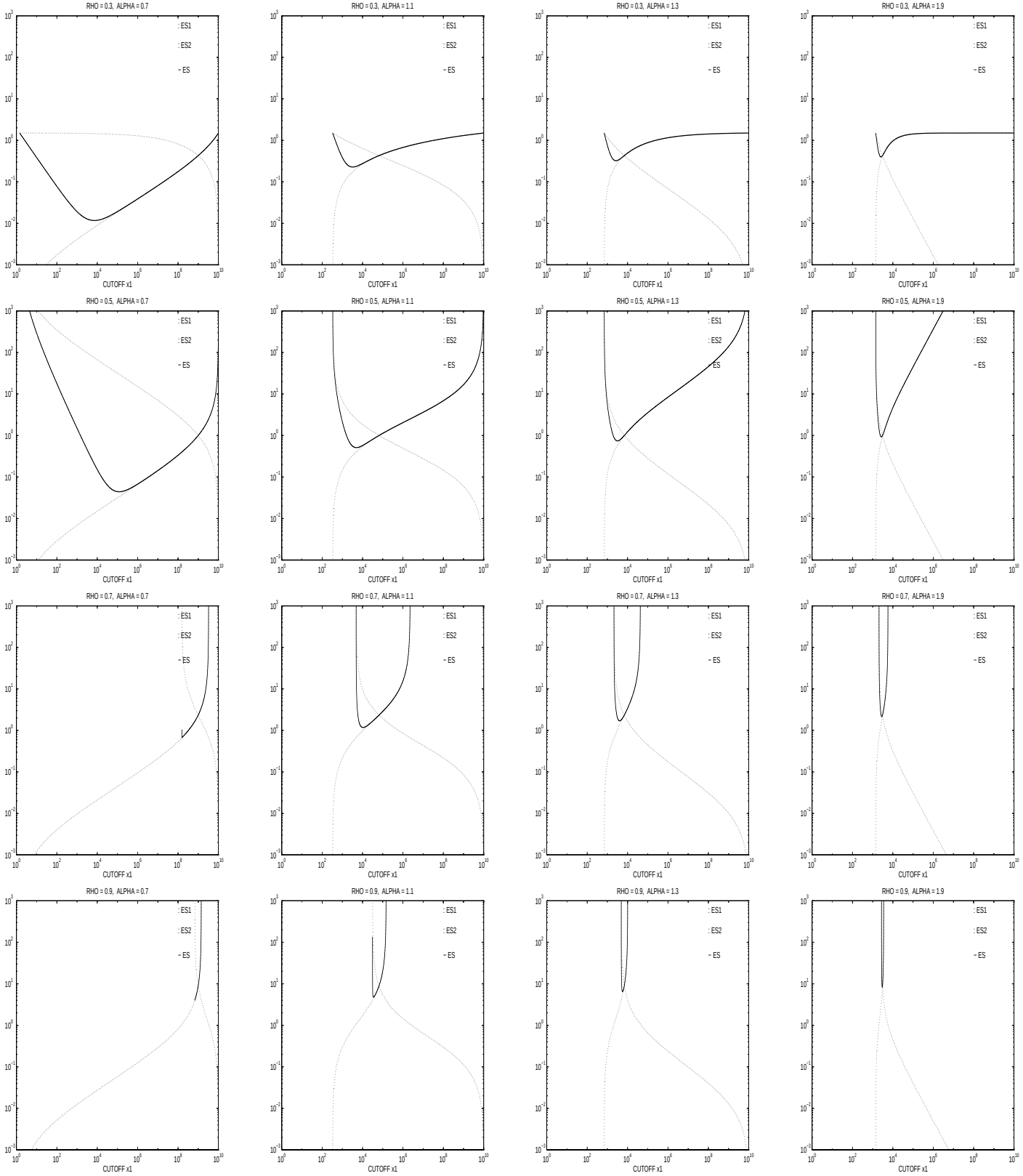


Figure 4: Performance of SITA-V_2 shown as a function of the cutoff, x_1, for α and ρ. The point where the two dotted curves cross represents $\mathbf{E}\left\{S^{EQ-LOAD}\right\}$. The minimum value for the sold line represents the slowdown for SITA-V_2, which is always an improvement over EQ-LOAD.

slowdown drops as ρ is increased. This effect is most pronounced when $\rho > .5$. The reason is that for $\rho > .5$, $\mathbf{E}\{S\}$ is only defined within the *legal region*—the region in which the load at both hosts is less than 1. This legal region lies between the vertical asymptotes of the $\mathbf{E}\{S_1\}$ and $\mathbf{E}\{S_2\}$ curves. As ρ increases, the legal region gets narrower, so there is less flexibility for the value of x_1. The problem is that if when x_1 approaches the limit of the narrow legal region, $\mathbf{E}\{S_2\}$ explodes. This means that $\mathbf{E}\{S\}$ has no alternative but to explode as well, regardless of how high p_1 is.

Luckily, this observation concerning high ρ is largely specific to the 2-host case. In Section 5 we will show that it is possible to compensate for the reduced flexibility of high load by increasing the number of hosts (h).

4.2 SITA-V for More Than 2 Hosts

For SITA-V_2, the value of x_1 that minimizes $\mathbf{E}\{S\}$ can be found in a straightforward way, since it is the inflection point of the $\mathbf{E}\{S\}$ curve (*i.e.,* the unique point within the legal region at which $d\mathbf{E}\{S\}/dx_1 = 0$).

The SITA-V algorithm for $h > 2$ hosts requires determining $h-1$ cutoff points, x_i, $i = 1 \ldots h-1$, such that $k = x_0 < x_1 < x_2 < \ldots < x_{h-1} < x_h = p$. All tasks of size between x_{i-1} and x_i are assigned to host i.

There are a number of ways to extend SITA-V_2 to the case $h > 2$. We have explored three methods:

Increasing i. In this approach, the x_is are determined in increasing order of i. That is, we first decompose the system into a single lightly-loaded host (number 1) and a "virtual" host with processing power equal to the remaining $h-1$ hosts. This determines x_1, and then the process is repeated recursively on tasks between size x_1 and x_h.

Decreasing i. This approach is just like the previous one, except that the heaviest-loaded host is considered first.

Divide-and-Conquer. This approach first finds the x_i corresponding to the division of the system into two (nearly) equal-sized subsystems; starting from lower bound x_0 and upper bound x_h, determine $x_{\lfloor h/2 \rfloor}$. The first group of $\lfloor h/2 \rfloor$ hosts are assigned all tasks of size between x_0 and $x_{\lfloor h/2 \rfloor}$ and the remaining group of $\lceil h/2 \rceil$ hosts receive all tasks of size between $x_{\lfloor h/2 \rfloor}$ and x_h. Then this process is repeated recursively within each of the two subsystems.

Although SITA-V_2 is by definition the optimal policy with respect to minimizing $\mathbf{E}\{S\}$, we have not shown that any of the three polices above are optimal for $h > 2$. Nonetheless, we find that in practice the three approaches described above typically provide similar results. Furthermore, their results are often quite good, as will be shown in the next section. As a result, for the remainder of the paper we present results obtained using just one policy: divide-and-conquer.

5 Evaluating SITA-V

In the previous section we showed how it is possible for SITA-V to reduce mean slowdown in a distributed system with heavy-tailed task sizes. However, although SITA-V reduces mean slowdown, it also increases mean waiting time $\mathbf{E}\{W\}$ in such a system—which is to be expected since minimal mean waiting time only occurs when load is balanced.

Therefore it is important to understand the tradeoff between decreases in slowdown and increases in waiting time in a system employing SITA-V. In this section we explore that tradeoff, by examining the performance metrics $\mathbf{E}\{S\}$ and $\mathbf{E}\{W\}$ for SITA-V over the wide range of values of α, ρ, and h shown in Table 1.

Figures 5 and 6 show the performance of SITA-V for α ranging from very high variability (0.5) to low variability (2.0), and h ranging from 2 to 16 in powers of two. Figure 5 shows results for conditions of high load ($0.6 \leq \rho \leq 0.9$), while Figure 6 shows results for conditions of low load ($0.2 \leq \rho \leq 0.5$). The left column in these figures indicates the improvement factor of SITA-V over an EQ-LOAD policy with respect to mean slowdown, that is, $\mathbf{E}\left\{S^{EQ-LOAD}\right\}/\mathbf{E}\left\{S^{SITA-V}\right\}$. The right column in these figures indicates the corresponding relative increase in waiting time: $\mathbf{E}\left\{W^{SITA-V}\right\}/\mathbf{E}\left\{W^{EQ-LOAD}\right\}$. Note that all of these plots use a log scale on the y axis, and that scales across all plots are the same.

We begin by restricting our attention to the left column of Figures 5 and 6, which show the improvement factor in mean slowdown. Several trends are clear: First, observe that regardless of the load, *the lower the α value, the greater the improvement in slowdown.* When $\alpha > 1.2$, performance improvements under SITA-V are not very large (between 1 and 2), but as α approaches 0.5, the factor improvement in slowdown rises to between 10^3 and 10^5 depending on load. To understand this effect of α, we return to a fundamental property of heavy-tailed distributions. Heavy-tailed task size distributions have the property that while the overwhelming majority of tasks are very small, more than half of the load is made up a tiny minority of the very largest tasks. SITA-V exploits heavy-tailed task size distributions by running the overwhelming majority on hosts which are loaded below the average system load, while running the tiny minority on hosts which are loaded above the average system load. Since mean slowdown is a per-task average, and most tasks are doing better, mean slowdown decreases. The smaller the α, the greater the size of the overwhelming majority, and thus the greater the improvement on mean slowdown.

Given this sensitivity to α, we can immediately ask how SITA-V would affect systems under empirically measured workloads. Empirical measurements of α vary. Our results show that for α values that seem to be at the high end of the range measured for Web servers ($\alpha \approx 1.1$) [6, 7], the improvement factor when using SITA-V (on more than 4 hosts) is around 2.5, across a wide range of loads. However, if file sizes in the Web have lower α values, as some measurements indicate [2], then SITA-V could exhibit much larger speedup improvements in practice.

Next we consider the effect of the load, ρ, and the number of hosts, h, on the mean slowdown improvement. The Figures show that decreasing the load or increasing the number of hosts results in an increase in mean slowdown improvement. The reason is that when the load is high, SITA-V has less flexibility for shifting around tasks (given the restriction that the load at all hosts must

stay below 1). Increasing the number of hosts increases this flexibility. This effect can be seen most clearly in the left-hand graphs in Figure 5. The uppermost of these graphs shows the case when $\rho = 0.9$ — extremely high load. In that case, the improvements possible under small α are not achieved until the number of hosts is large ($h = 16$). Moving to the next graph downward, in which ρ has decreased to 0.8, we see that the maximum improvements are now achieved at a lower number of hosts ($h = 8$). Progressing downward again, we see that this "knee" continues to move toward lower h values. Thus we can conclude that when α is small, significant improvements are possible if *either* ρ is small, or h is large.

Now we turn to the tradeoff issues represented by the factor increases in waiting time shown in the right-hand columns of Figures 5 and 6). First consider the effect of α. Similarly to slowdown, the increase in waiting time goes up as α goes down. However, this increase tapers off for the lower α (below $\alpha = .9$), whereas the improvement in slowdown continues to increase as α is dropped. Next consider the effect of ρ. The load, ρ, effects the size of the range on the y-axis as we move from $\alpha = 2.0$ to $\alpha = .5$. For high ρ, the factor increase in waiting time varies from around 1 to around 600 as α varies. For low ρ ($\rho = .2$) this range shrinks to 1 to 20. These effects combine so that SITA-V appears especially attractive for low ρ and low α where its factor improvement in mean slowdown is on the order of 10^5 whereas its factor increase in mean waiting time is only on the order of 20.

The tradeoff between the improvement in slowdown and the increase in waiting time is summarized in Figure 7. This figure plots relative decrease in slowdown and relative increase in waiting time for four representative cases from the range of ρ and α; note that in this figure the y-axes are not all to the same scale.

For high α ($\alpha = 1.5$), the figure shows that only modest improvements in slowdown are possible, and that the corresponding costs in waiting time are also small; this holds regardless of the load ρ. However, for low α ($\alpha = 0.5$) we can distinguish two cases based on the value of ρ. For $\rho = 0.2$, significant improvements in slowdown are possible, while waiting time is not increased by the same factor. When load on the system is high ($\rho = 0.8$), the cost in increased waiting time in using SITA-V is also high; however the improvements in slowdown due to SITA-V can also be high, if enough hosts are used ($h > 8$).

In summary we note that SITA-V and EQ-LOAD represent ends of a spectrum. SITA-V minimizes slowdown while EQ-LOAD minimizes waiting time. These two metrics correspond respectively to optimizing the per-file case, and the per-byte case (as discussed in Section 2). Thus, these results may be interpreted as an indication of how far apart these two operating points are when workloads are heavy tailed. This suggests that there may be more desirable policies that operate the system at an intermediate point between these two extremes.

6 Using SITA-V in a Distributed Web Server

One of the most common bottlenecks in the Web is the performance of servers. To increase capacity, builders of high performance Web servers are increasingly turning to distributed systems because of their potential support for scalability [16, 12, 20, 24]. For example, Netscape's home site has

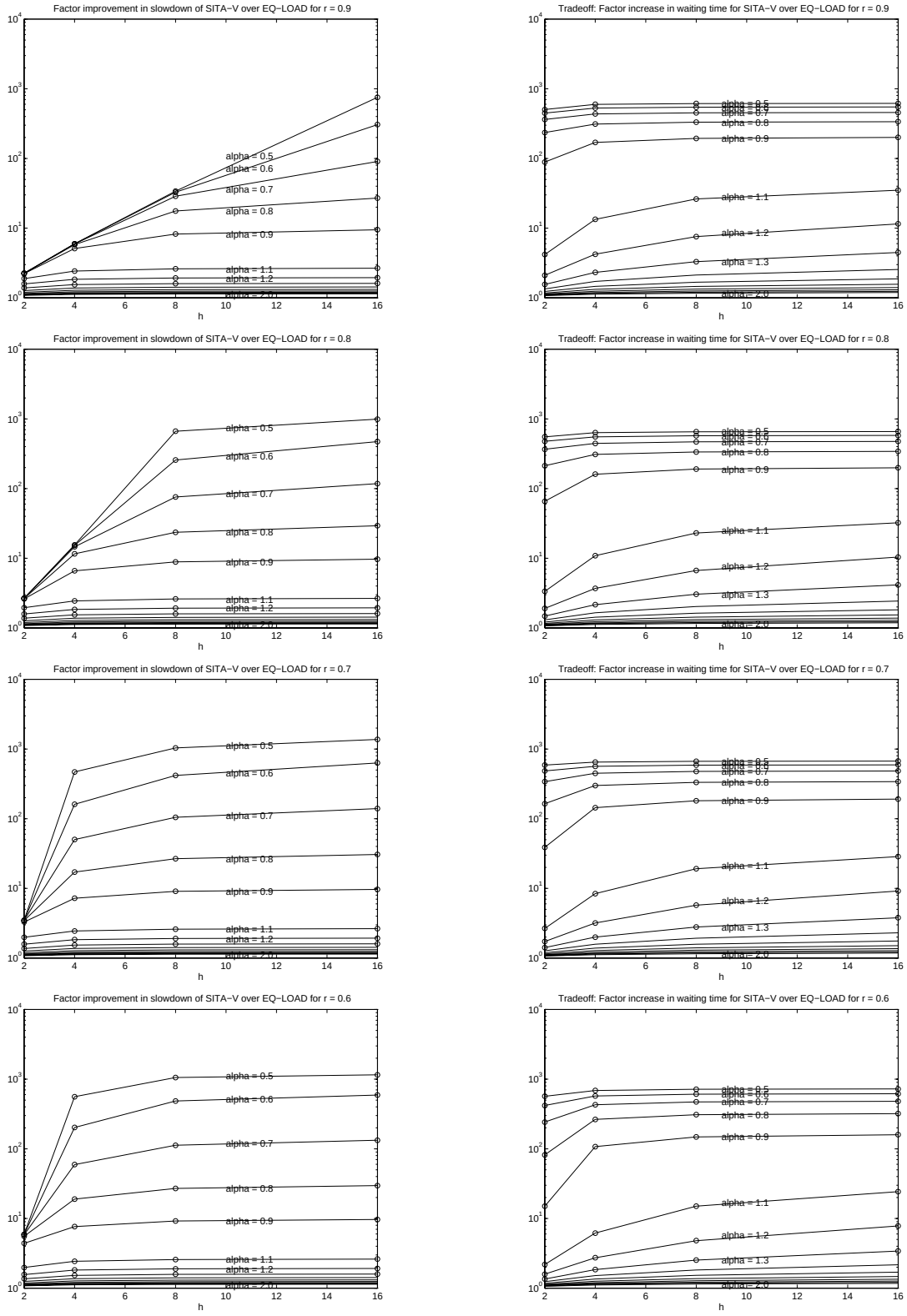


Figure 5: Big ρs, all α, all h. Left column shows improvement factor for slowdown. Right column shows tradeoff factor for waiting time.

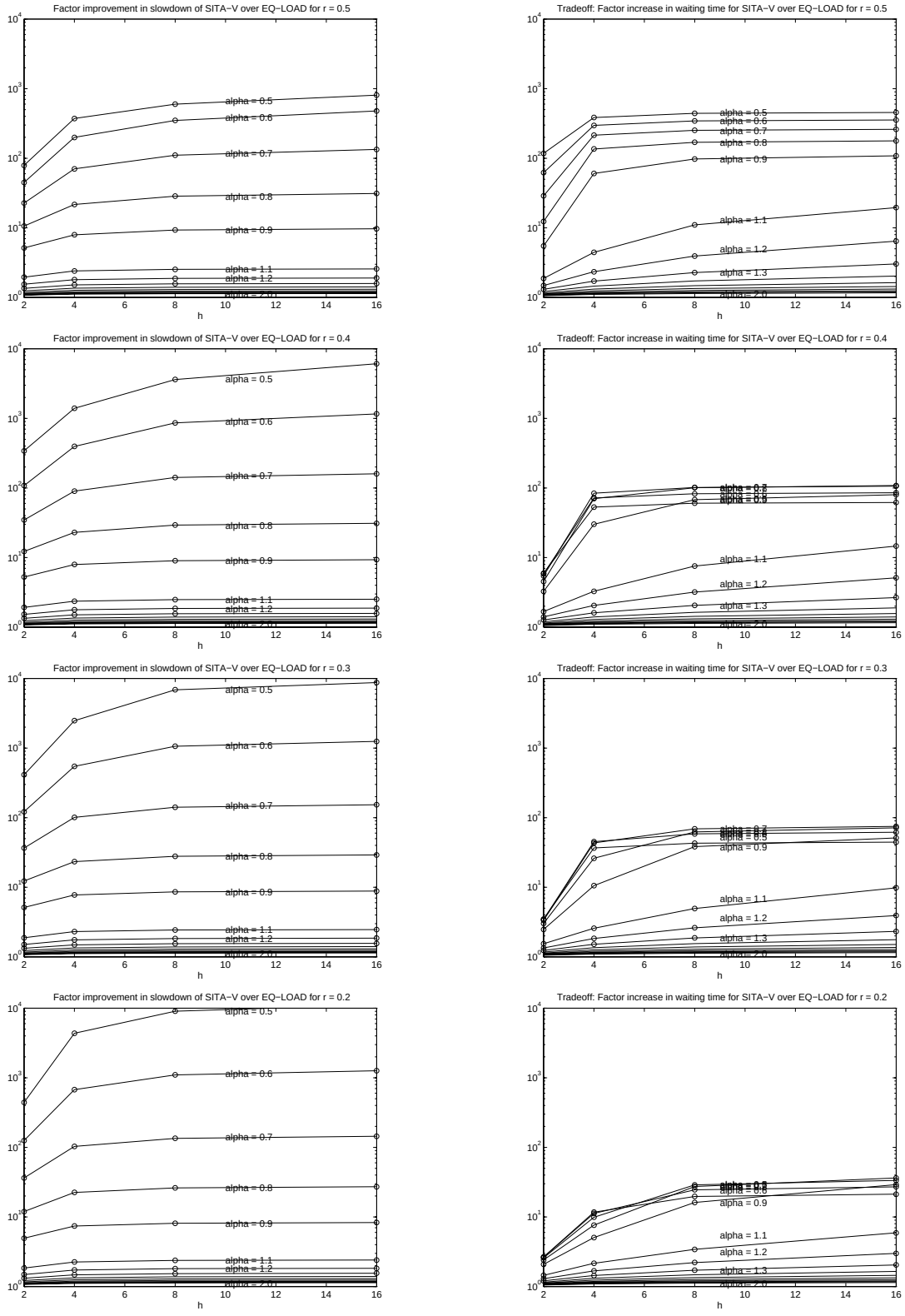


Figure 6: Small ρs, all α, all h. Left column shows improvement factor for slowdown. Right column shows tradeoff factor in waiting time.

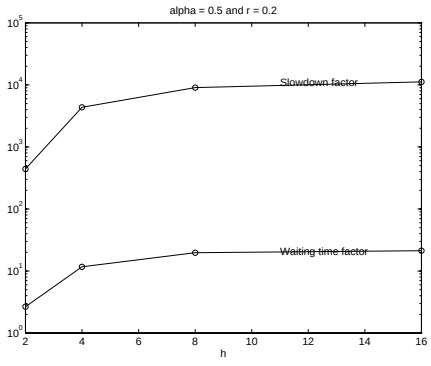
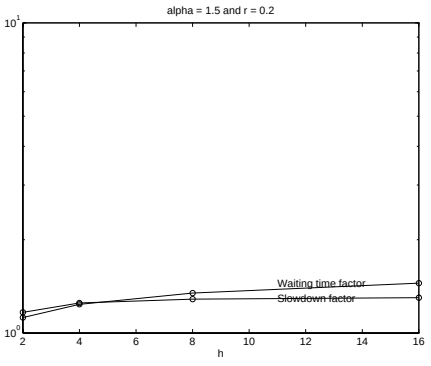
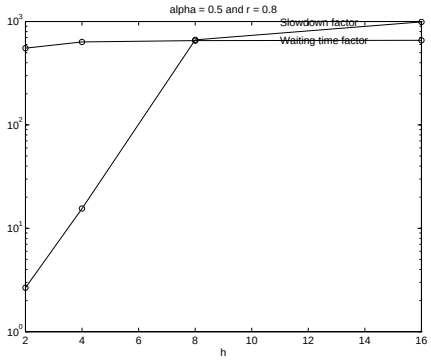
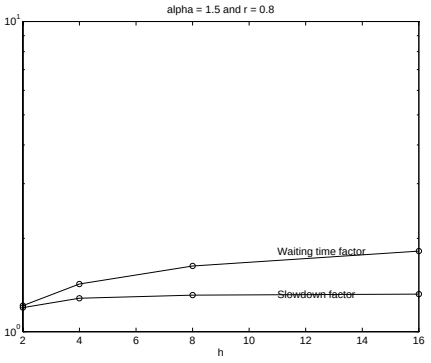
	$\alpha = 0.5$	$\alpha = 1.5$
$\rho = 0.2$		
$\rho = 0.8$		

Figure 7: Tradeoff Regions for SITA-V.

evolved from a 6-processor tightly-coupled multiprocessor serving about 5 million hits per day, to a distributed system with more than 20 hosts serving more than 100 million hits per day [12, 25, 26].

Recent performance measurements of the Web and Web servers show that Web file sizes exhibit heavy-tailed distributions, leading to very high variability [2, 6]. As a result, practical resource allocation methods for Web servers must address the basic problems that arise from highly variable file sizes.

In this section we review the state of the art in distributed Web servers, and show that task assignment is an important problem in such systems; we show that mechanisms for centralized task assignment are commercially available; and we suggest how SITA-V could be useful in such systems.

The most commonly used task-assignment mechanism in current distributed Web servers employs a feature of the Domain Name System (DNS) called Round-Robin DNS [4, 16]; this approach attempts to balance load among hosts by equalizing the number of tasks assigned to each host. However, the time scale at which RR-DNS balances load is relatively coarse, which leads to significant load imbalance in practice [5, 12, 23]. This has motivated the development of fine-grained load balancing mechanisms, such as the TCP router. The TCP router modifies the destination IP address within each packet in such a way that all packets stemming from one request go to the same server host, but successive requests are mapped to successive hosts in a round-robin fashion; this allows equality in work assignment at a very fine time scale. TCP routers have been employed in experimental Web servers [1, 10, 22]. Such routers may introduce delays because they must modify each packet that flows through them. However, because they achieve better load balance, they are commonly proposed for high-performance distributed Web servers [10, 22], and a number of commercial products have appeared that implement this functionality.

A recent enhancement of TCP routers, also aimed at better load balance, is dynamic task assignment based on information about the current load on each host. In this approach, hosts update the task router constantly on their load status, and the task router sends an incoming task to the least-loaded host. This method is used in commercial products such as IBM's Network Dispatcher, Cisco's Local Director, F5 Labs's Big/IP, RND Network's Web Server Director and others. These products are called "load balancers" by popular magazines and their task assignment schemes have received much recent attention, [3, 30]. However using a dynamic task assignment scheme has two serious drawbacks: first, the large amount of information collection at a single point (the router) is an impediment to scaling; and second, the load data received by the router is often stale, resulting in tasks being mistakenly routed to a server host that is already overloaded.

Regardless of the approach used, most current methods attempt to equalize load across hosts. In this paper we have shown that, in contrast to systems with low variability in task size, the high variability in task sizes present in the Web suggests that assigning equal amounts of work to each host of a distributed Web server is not necessarily the best policy.

Implementation of SITA-V in a TCP router requires that the task assignment policy can determine a task's service demand upon its arrival to the system. We assume that a task's service demand can be inferred from the name of the file being requested, which is reasonable for static data. The implementation of a TCP router that can inspect file names before forwarding requests to hosts is described, for example, in [14].

In addition to improving performance, SITA-V has practical benefits as well. No data collection from the hosts is required, so neither of the problems listed above associated with dynamic task assignment schemes are an issue. An additional design benefit of SITA-V is that it doesn't require duplicating all the files onto every host; each host is only responsible for a subset of all files.

7 Related Work

There is a huge body of literature on load balancing in general distributed systems, including analytic, simulation, and implementation work (see [13] for a number of references). More recently, analytic work on load balancing has considered more general task size distributions than the traditional exponential distribution (this usually requires some decomposition approximation where the nodes of the network are assumed to behave independently of each other, see for example, [9]). However the specific properties of *heavy-tailed* task sizes are only just now beginning to be incorporated into the design of load balancing policies [13]. Performance analysis of load balancing policies under heavy-tailed task sizes is usually difficult. Heavy-tailed distributions have however been considered in some load balancing simulations [21, 18, 13].

The previous section described related work in the context of distributed Web servers. Previous Web server load balancing research has not considered the effects of a heavy-tailed task size distribution. For example, the work described in [5] has looked at load balancing in distributed Web servers but has concentrated on developing coarse-grained load balancing policies for use within DNS; variability in request sizes is not considered.

Nonetheless, heavy-tailed distributions are important because they are beginning to appear in many measurements of computer systems. Work in [6, 7, 2] has shown that Web file sizes often exhibit heavy tails. The ranges of α reported in [6, 7] are approximately 1.1 to 1.3. There is evidence that file sizes in systems other than the Web may show heavy tails as well: Unix file size measurements are presented in [15], and I/O in a general computing enviroment is presented in [28]. Also, the authors in [27] found that the upper tail of the distribution of data bytes in FTP bursts (file transfers over the Internet) was well fit to a heavy-tailed distribution with $0.9 \leq \alpha \leq 1.1$.

In addition to files and I/O traces, other computer system attributes have also been shown to exhibit heavy tailed distributions. In particular, the lifetime of processes in some systems can show heavy tails: [21] found that Unix process lifetimes showed heavy tails with $1.05 \leq \alpha \leq 1.25$ and [13] found that Unix process lifetimes showed heavy tails with $\alpha \approx 1$.

Analysis of heavy-tailed distributions has been developed only recently in the area of queuing theory. All has been in the context of a single queue. Analysis of a M/G/1 queue with heavy-tailed task service times has appeared in [29]. Analysis of queues with infinite variance task sizes is difficult. For this reason finite variance approximations to heavy-tailed distributions are increasingly being used [11].

8 Conclusion

In this paper we've shown that for distributed systems with heavy-tailed workloads employing a processor-sharing policy at the hosts, mean slowdown can be improved by large factors by adopting a policy that does not balance load. We have introduced a new policy, SITA-V, that reduces mean slowdown to levels far below those that a balanced-load policy would achieve. The key idea of SITA-V is to direct the small but numerous tasks to lightly-loaded hosts, while sending the large but rare tasks to heavily-loaded hosts.

We have argued that slowdown is an important metric because it relates a task's waiting time to its service demand. In a system with highly variable task sizes, users may be willing to wait longer for large tasks to complete, but expect that small tasks should complete quickly. Reducing mean waiting time does not by itself insure that this will be the case.

In addition, we have suggested that SITA-V may have applicability in the context of modern distributed Web servers. We showed that modern distributed Web servers all assume that load should be balanced among the server hosts, in contrast to the SITA-V strategy. In addition we showed that implementing SITA-V is feasible given current distributed Web server technology, and that it has design advantages over currently used task assignment policies.

The cost in employing SITA-V comes in increased mean waiting time. We have evaluated the tradeoff between reducing slowdown and increasing waiting time in a system employing SITA-V. The two factors that determine the nature of the tradeoff are the variability of tasks (as measured by the exponent α in the task size distribution) and the overall system utilization ρ. In general, we find that when task sizes are highly variable ($\alpha < 1$) SITA-V can result in remarkable improvements in slowdown — by factors as great as 1000 or more. For task size distributions similar to those found on the Web, ($\alpha \approx 1.1$), the improvement is on the order of a factor of 2.5. For high ρ this improvement is possible using only a small number of hosts (4), however for large ρ more hosts (16) are needed.

These results suggest that the variability present in heavy-tailed workloads is so high as to suggest that some traditional issues in system design need to be reconsidered. For example, a common rule of thumb among computer systems designers is to "optimize the common case" [19]. However when task sizes follow a heavy-tailed distribution, an important question becomes: what is the common case? Consider a set of files in which most files are small, but half of the bytes are contained in the 0.3% largest files. Then there are two answers to this question: there is a "common file," (*i.e.,* a small file) and there is a "common byte" (*i.e.,* those in large files). We show in this paper that these two answers to the common case question lead to radically different system designs in practice. As discussed at the end of Section 2, waiting time can be considered to be equivalent to per-byte slowdown. So we can observe that a system that focuses on good per-file performance (that is, one that minimizes slowdown) is remarkably different from a system that focuses on good per-byte performance (that is, one that minimizes waiting time). We conclude that the nature of heavy-tailed workloads places the per-byte metric and the per-file metric in strong opposition.

References

[1] E. Anderson, D. Patterson, and E. Brewer. The magicrouter: An application of fast packet interposing. Technical report, UC Berkeley, 1996.

[2] Martin F. Arlitt and Carey L. Williamson. Web server workload characterization: The search for invariants. In *Proceedings of the 1996 SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 126–137, 1996.

[3] Padraic Boyle. Web site traffic cops: Load balancers can provide the busiest web sites with non-stop performance. *PC Magazine. Available online at http://www8.zdnet.com/pcmag/*, February 18, 1997.

[4] T. Brisco. DNS support for load balancing. RFC 1794, USC/Information Sciences Institute. Available at ftp://ds.internic.net/rfc/rfc1794.txt., April 1995.

[5] Michele Colajanni, Philip S. Yu, and Daniel M. Dias. Scheduling algorithms for distributed web servers. In *Proceedings of ICDCS '97*, 1997.

[6] Mark E. Crovella and Azer Bestavros. Self-similarity in World Wide Web traffic: Evidence and possible causes. In *Proceedings of the 1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 160–169, May 1996.

[7] Mark E. Crovella, Murad S. Taqqu, and Azer Bestavros. Heavy-tailed probability distributions in the world wide web. In *A Practical Guide To Heavy Tails*, chapter 1, pages 1–23. Chapman & Hall, New York, 1998.

[8] Carlos A. Cunha, Azer Bestavros, and Mark E. Crovella. Characteristics of WWW client-based traces. Technical Report TR-95-010, Boston University Department of Computer Science, April 1995.

[9] E. Lazowska D. Eager and J. Zahorjan. Aadaptive load sharing in homogeneous distributed systems. *IEEE Transactions on Software Engineering*, 12(5), 1986.

[10] D.M.Dias, W. Kish, R. Mukherjee, and R. Tewari. A scalable and highly available Web server. In *Proceedings of the 41st IEEE Computer Society International Conference (COMPCON) '96*, pages 85–92, February 1996.

[11] Anja Feldmann and Ward Whitt. Fitting mixtures of exponentials to long-tail distributions to analyze network performance models. In *Proceedings of IEEE INFOCOM'97*, pages 1098–1116, April 1997.

[12] Simson L. Garfinkel. The wizard of Netscape. *WebServer Magazine*, 1(2):59–63, 1996.

[13] Mor Harchol-Balter and Allen Downey. Exploiting process lifetime distributions for dynamic load balancing. In *Proceedings of SIGMETRICS '96*, pages 13–24, 1996.

[14] Guerney Hunt, Erich Nahum, and John Tracey. Enabling content-based load distribution for scalable services. Preprint, 1997.

[15] Gordon Irlam. Unix file size survey - 1993. Available at `http://www.base.com/gordoni-/ufs93.html`, September 1994.

[16] E. D. Katz, M. Butler, and R. E. McGrath. A scalable web server: The NCSA prototype. In *Proceedings of the First International WWW Conference*, 1994.

[17] Leonard Kleinrock. *Queueing Systems*, volume II. Computer Applications. John Wiley & Sons, 1976.

[18] Phillip Krueger and Miron Livny. A comparison of preemptive and non-preemptive load distributing. In *8th International Conference on Distributed Computing Systems*, pages 123–130, June 1988.

[19] Butler W. Lampson. Hints for computer system design. *Proceedings of the Ninth SOSP, in Operating Systems Review*, 17(5):33–48, October 1983.

[20] K.L.E. Law, B. Nandy, and A. Chapman. A scalable and distributed WWW proxy system. In *Proceedings of ACM Multimedia '97*, 1997.

[21] W. E. Leland and T. J. Ott. Load-balancing heuristics and process behavior. In *Proceedings of Performance and ACM Sigmetrics*, pages 54–69, 1986.

[22] Y.-H. Liu, P. Danzig, C.E. Wu, J. Challenger, and L.M. Ni. A distributed Web server and its performance analysis on multiple platforms. In *Proceedings of the 16th Internation Conference on Distributed Computing Systems (ICDCS)*, pages 665–672, 1996.

[23] Jeffrey C. Mogul. Network behavior of a busy web server and its clients. Technical report, DEC WRL Research Report 95/5, October 1995.

[24] Antoine Morad and Huiqun Liu. Redirection-based scalable web server architecture. preprint, 1997.

[25] Dan Mosedale, William Foss, and Rob McCool. Administering very high volume internet services. Available at http://www.keynote.com/techrpts/nspaper.html.

[26] Dan Mosedale, William Foss, and Rob McCool. Lessons learned administering netscape's internet site. *Internet Computing*, 1(2):28–35, 1997.

[27] Vern Paxson and Sally Floyd. Wide-area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking*, pages 226–244, June 1995.

[28] David L. Peterson. Data center I/O patterns and power laws. In *CMG Proceedings*, December 1996.

[29] E. Willekens and J.L. Teugels. Asymptotic expansions for waiting time probabilities in an M/G/1 queue with long-tailed service times. *Queueing Systems*, 10:295–312, 1992.

[30] Greg Yerxa. Web server redirectors balance your web load. *Network Computing – CMPnet. Available online at http://techweb.cmp.com/nc/814/814cn2.html*, July 31, 1997.