On Choosing a Task Assignment Policy for a Distributed Server System

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

Mark E. Crovella[†] Cristina D. Murta[‡]
Department of Computer Science
Boston University
Boston, MA 02215
{crovella,murta}@bu.edu

Abstract

We consider a distributed server system model and ask which policy should be used for assigning tasks to hosts. In our model each host processes tasks in First-Come-First-Serve order and the task's service demand is known in advance. We consider four task assignment policies commonly proposed for such distributed server systems: Round-Robin, Random, Size-Based, in which all tasks within a give size range are assigned to a particular host, and Dynamic-Least-Work-Remaining, in which a task is assigned to the host with the least outstanding work. Our goal is to understand the influence of task size variability on the decision of which task assignment policy is best. We evaluate the above policies using both analysis and simulation. We find that no *one* of the above task assignment policies is best and that the answer depends critically on the variability in the task size distribution. In particular we find that when the task sizes are not highly variable, the Dynamic policy is preferable. However when task sizes show the degree of variability more characteristic of empirically measured computer workloads, the Size-Based policy is the best choice. We use the resulting observations to argue in favor of a specific size-based policy, SITA-E, that can outperform the Dynamic policy by almost 2 orders of magnitude and can outperform other task assignment policies by many orders of magnitude, under a realistic task size distribution.

Keywords: Task assignment, scheduling, heavy-tailed workloads, high variance, distributed servers, queueing theory.

[*] 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.

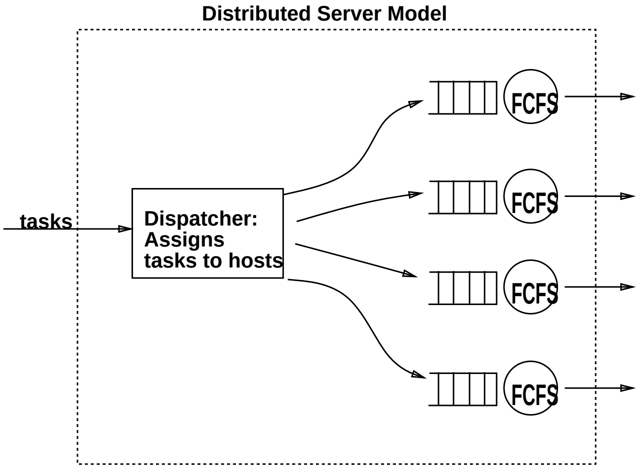


Figure 1: Distributed Server Model

1 Introduction

To build high-capacity server systems, developers are increasingly turning to distributed designs because of their scalability and cost-effectiveness. Examples of this trend include distributed Web servers, distributed database servers, and high performance computing clusters.

In such a system, requests for service arrive and must be assigned to one of the host machines for processing. The rule for assigning tasks to host machines is known as the *task assignment policy*. The choice of the task assignment policy has a significant effect on the performance perceived by users, and the question of which task assignment policy is "best" has been asked repeatedly in various contexts and still remains open for many models.

In this paper we concentrate on the particular model of a distributed server system in which each incoming task is immediately assigned to a host machine, and each host machine processes its assigned tasks in first-come-first-served (FCFS) order. Such a system is shown in Figure 1. We also assume that the task's service demand is known in advance. Our motivation for considering this model is that it is an abstraction of some existing distributed servers, described in Section 3.

We consider four task assignment policies commonly proposed for such distributed server systems: Round-Robin assignment, Random assignment, Size-Based assignment, and Dynamic assignment (also known as Least-Work-Remaining assignment). In Round-Robin assignment tasks are assigned to hosts in a cyclical fashion. In Random assignment each task is assigned to each host with equal probability. In Size-Based assignment all tasks within a certain size range are sent to an individual host. In Dynamic assignment an incoming task is assigned to the host with the least amount of outstanding work left to do (based on the sum of the sizes of those tasks in the queue). We also briefly discuss some other task assignment policies, like Shortest-Line assignment, for comparison purposes.

Our goal is to study the influence of task size variability on the decision of which task assignment policy is best. We are motivated in this respect by the increasing evidence for high variability in task size distributions, as seen in many measurements of computer workloads. In particular, measurements of many computer workloads have been shown to fit a heavy-tailed distributions with very high variance, as described in Section 2.2. A heavy-tailed distribution is one whose tail declines like a power-law, that is $\Pr\{X > x\} \sim x^{-\alpha}$ for $0 < \alpha \leq 2$. By varying α we can study distributions that show moderate variability ($\alpha \approx 2$) to high variability ($\alpha \approx 1$).

In comparing task assignment policies, we make use of simulations and also analysis or analytic approximations. We show that the variability of the task size distribution makes a crucial difference in choosing a task assignment policy, and we use the resulting observations to argue for a specific task assignment policy that works well under conditions of high task size variance.

2 Background and Previous Work

This section is divided into two parts. In Section 2.1 we briefly review some fundamental results from the literature on optimality of task assignment policies. In Section 2.2 we review measurements of the task size distribution in various computer applications.

2.1 Fundamental Results in Task Assignment

The problem of task assignment in a model like ours has been extensively studied, but many basic questions remain open. In the case where task sizes are unknown, the following results exist: Under an exponential task size distribution, the optimality of Shortest-Line task assignment policy (send the task to the host with the shortest queue) has been proven by Winston [15]. This optimality result was extended by Weber [13] to include task size distributions with nondecreasing failure rate. The actual performance of the Shortest-Line policy is not known exactly, but is approximated by Nelson and Phillips [8]. These approximations have been extended to include more general task size distributions than the exponential distribution, however the approximations grow less accurate as the variability of the task size distribution increases [9]. In fact as the variability of the task size distribution grows, the Shortest-Line policy is no longer optimal, as proven by Whitt [14].

In the case where the individual task sizes are known, as in our model, equivalent optimality and performance results have not been developed, to the best of our knowledge. The scenario has been considered in which the ages (time in service) of the tasks currently serving are known, so that it is possible to compute an arriving task's expected delay at each queue. In this scenario, Weber [13] has shown that the Shortest-Expected-Delay rule is optimal for task size distributions with increasing failure rate, and Whitt [14] has shown that there exist task size distributions for which the Shortest-Expected-Delay rule is not optimal.

2.2 Measurements of task size distributions in computer applications

As described in Section 1, we are concerned with how the distribution of task sizes affects the decision of which task assignment policy to use.

Many application environments show a mixture of task sizes spanning many orders of magnitude. In such environments there are typically many small tasks, and fewer large tasks. Much previous work has used the exponential distribution to capture this variability, as described in Section 2.1. However, recent measurements indicate that for many applications the exponential distribution is a poor model and that a heavy-tailed distribution is more accurate. In general a heavy-tailed distribution is one for which

$$\Pr\{X > x\} \sim x^{-\alpha},$$

where $0 < \alpha < 2$. 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

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

A set of task sizes following a heavy-tailed distribution has the following properties:

1. Decreasing failure rate: In particular, the longer a task has run, the longer it is expected to continue running.

2. Infinite variance (and if $\alpha \leq 1$, infinite mean).

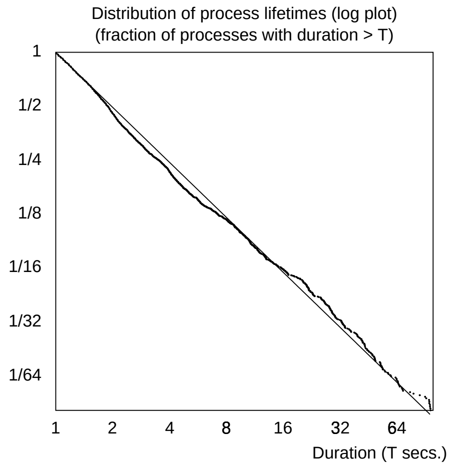


Figure 2: Measured distribution of UNIX process CPU lifetimes, taken from [HD97]. Data indicates fraction of jobs whose CPU sevice demands exceed T seconds, as a function of T.

3. The property that a very small fraction ($< 1\%$) of the very largest tasks make up a large fraction (half) of the load. We will refer to this important property throughout the paper as the *heavy-tailed property.*

The lower the parameter α, the more variable the distribution, and the more pronounced is the heavy-tailed property, *i.e.* the smaller the faction of large tasks that comprise half the load.

As a concrete example, Figure 2 depicts graphically on a log-log plot the measured distribution of CPU requirements of over a million UNIX processes, taken from paper [4]. This distribution closely fits the curve

$$\Pr\{\text{Process Lifetime } > T\} = 1/T.$$

In [4] it is shown that this distribution is present in a variety of computing environments, including instructional, reasearch, and administrative environments.

In fact, heavy-tailed distributions appear to fit many recent measurements of computing systems. These include, for example:

- Unix process CPU requirements measured at Bellcore: $1 \leq \alpha \leq 1.25$ [7].
- Unix process CPU requirements, measured at UC Berkeley: $\alpha \approx 1$ [4].
- Sizes of files transferred through the Web: $1.1 \leq \alpha \leq 1.3$ [1, 3].
- Sizes of files stored in Unix filesystems: [6].

- I/O times: [11].
- Sizes of FTP transfers in the Internet: $.9 \leq \alpha \leq 1.1$ [10].

In most of these cases where estimates of α were made, $1 \leq \alpha \leq 2$. In fact, typically α tends to be close to 1, which represents very high variability in task service requirements.

3 Model and Problem Formulation

We are concerned with the model of a distributed server shown in Figure 1. The server is composed of h hosts, each with equal processing power. Tasks arrive to the system according to a Poisson process with rate λ. When a task arrives to the system, it is inspected by the dispatcher facility which assigns it to one of the hosts for service. We assume the dispatcher facility knows the size of the task. The set of tasks assigned to each host is served in FCFS order, and tasks are not preemptible. We assume that processing power is the only resource used by tasks (*e.g.,* we do not model I/O demand of a task). Note that in this model, the hosts themselves may be multiprocessor machines.

The above model for a distributed server was initially inspired by the `xolas` distributed computing facility at MIT's Laboratory for Computer Science. `Xolas` is a distributed batch computing server which processes computationally-intensive, possibly parallel jobs which are submitted from various departments throughout the campus. `Xolas` consists of 4 identical multiprocessor hosts. Users submit jobs to `xolas` and specify an upper bound on their job's processing demand. If the job exceeds that demand, it is killed. The `xolas` facility has a dispatcher front end which assigns each job to one of several identical hosts for service. The user is then informed of the queue to which her job has been assigned and is given an upper bound on the time the job will have to wait in the queue, based on the sum of the sizes of the jobs in that queue. The jobs queued at each host are serviced in FCFS order.

More generally, our assumption that task sizes are known holds to an approximate degree in some batch computing servers. This assumption is also reasonable in some other contexts, *e.g.,* HTTP requests for static web files. Usually the majority of HTTP requests received at a Web site are for static files (rather than "cgi" scripts). The sizes of all the files at the Web site can be known to the dispatcher, allowing the dispatcher to determine a task's size when the task arrives.[2]

The assumption that tasks are kept in queues at the hosts rather than a single central queue removes the need for global synchronization at the central queue.

The assumption that tasks are served in FCFS order is for the purpose of user satisfaction. In a batch computing facility, users don't want jobs arriving after them to affect their waiting time (users want to know an upper bound on their total waiting time).

Finally, the assumption that tasks are not preemptible is also often true in batch computing environments; it can be costly to perform context switching, and space-sharing policies are often preferable to time-sharing when the hosts themselves are parallel systems.

What remains to be defined in the above model for a distributed server is the distribution on task sizes that we assume. We assume that task sizes show some maximum (but large) value. Note that this would be the expected case for a file 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:

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

Throughout this paper we model task sizes using a $B(k, p, \alpha)$ distribution, and vary α over the range 0 to 2 in order to observe the effect of changing variability of the distribution. To focus on the effect of changing variance, we keep the distributional mean fixed (at 3000) and the maximum value fixed (at $p = 10^{10}$), which correspond to typical values taken from [1]. In order to keep the mean constant, we adjust k slightly as α changes ($0 < k \leq 1500$). The above parameters are summarized in Table 1.

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 very high variability if $k \ll p$. For example, Figure 3 (right) shows the second moment $\mathbf{E}\{X^2\}$ of this 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. Furthermore, the Bounded Pareto distribution also still exhibits the heavy-tailed

[2] Of course, there are many other issues involved in task assignment schemes for distributed web servers, which we are not modeling, such as network delays between the host and the client.

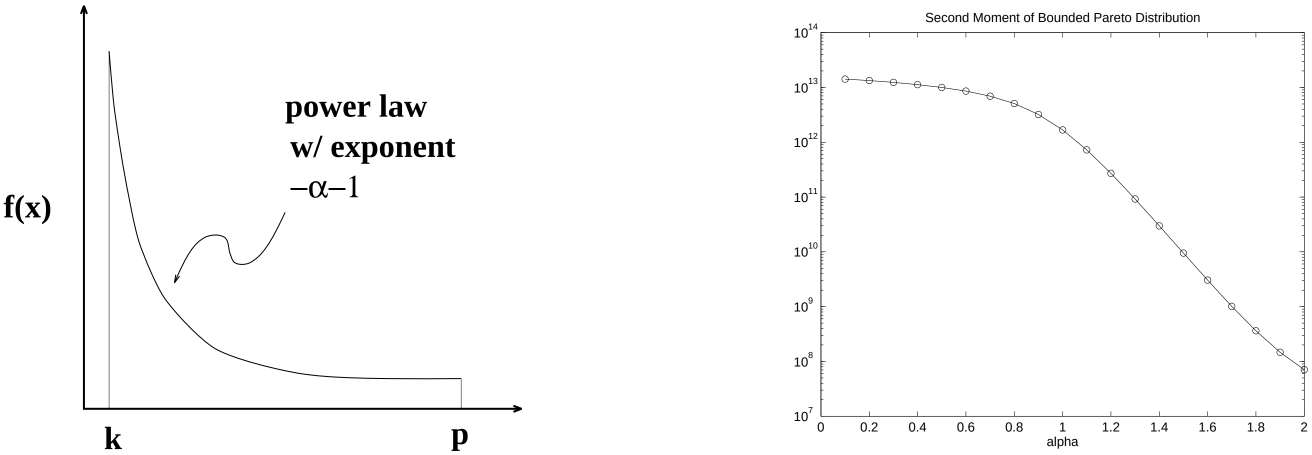


Figure 3: Parameters of the Bounded Pareto Distribution (left); Second Moment of $B(k, 10^{10}, \alpha)$ as a function of α, when $\mathbf{E}\{X\} = 3000$ (right).

Number of hosts	$h = 8$. Later consider increase.
System load	$\rho = .8$. Later consider range.
Mean service time	$\mathbf{E}\{X\} = 3000$ time units
Task arrival rate	$\lambda = \rho \cdot 1/\mathbf{E}\{X\} \cdot h = .0021$ tasks/unit time
Maximum task service time	$p = 10^{10}$ time units
α parameter	$0 < \alpha \leq 2$
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

property and (to some extent) the decreasing failure rate property of the unbounded Pareto distribution. We mention these properties because they are important in determining our choice of the best task assignment policy.

Given the above model of a distributed server system, we are concerned with the question of how to select the best task assignment policy. The following four task assignment policies are common choices:

Random : an incoming task is sent to host i with probability $1/h$. This policy equalizes the expected number of tasks at each host.

Round-Robin : tasks are assigned to hosts in cyclical fashion with the ith task being assigned to host i mod h. This policy also equalizes the expected number of tasks at each host, and typically has less variability in interarrival times than Random.

Size-Based : Each host serves tasks whose service demand falls in a designated range. For example, in a batch computing environment one might have a queue for tasks of 30 minutes or less, a queue for tasks 30 minutes to 2 hours, a queue for tasks greater than 2 hours. This policy attempts to keep small tasks from getting "stuck" behind large tasks.

Dynamic : Each incoming task is assigned to the host with the smallest amount of outstanding work, which is the sum of the sizes of the tasks in the host's queue plus the work remaining on that task currently being served. This policy is optimal from the standpoint of an individual task, and from a system standpoint attempts to achieve instantaneous load balance.

The reason that the last policy is called "Dynamic" is that the assignment decision uses knowledge of the state of the hosts. In contrast the first three policies are static; in those cases the assignment decision is made without knowledge of the state of the hosts.

The question we ask in this paper is which of these policies is best, and how they compare in terms of performance, as a function of the variability of task sizes. The effectiveness of these task assignment schemes will be measured in terms of mean waiting time, mean queue length, and mean slowdown, where a task's slowdown is its waiting time divided by its service demand. All means are per-task averages.

Of these three metrics, 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.

In analyzing the distributed system model presented here we use the notation in Table 2.

W	Waiting time for tasks arriving at the system	W_i	Waiting time for only those tasks at the ith host
S	Slowdown for tasks arriving at the system	S_i	Slowdown for only those tasks at the ith host
Q	Number of tasks an arriving task sees ahead of it	Q_i	Number of tasks at the ith host on task arrival
X	Service time (execution time) for tasks	X_i	Execution time for those tasks at the ith host
λ	Arrival rate into the system	λ_i	The arrival rate at the ith host
p_i	The probability that a task is sent to the ith host. (May not be defined for some policies)		

Table 2: Notation Used in the Paper

Before delving into simulation and analytic results, we need to specify a few more parameters of the size-based policy. We do that below in Section 3.1.

3.1 A New Size-Based Task Assignment Policy: SITA-E

In size-based task assignment, a size range is associated with each host and a task is sent to the appropriate host based on its size. In practice the size ranges associated with the hosts are often chosen somewhat arbitrarily. There might be a 15-minute queue for tasks of size between 0 and 15 minutes, a 3-hour queue for tasks of size between 15 minutes and 3 hours, a 6-hour queue, a 12-hour queue and an 18-hour queue, for example. (This example is used in practice at the Cornell Theory Center IBM SP2 job scheduler [5].)

In this paper we choose a more formal algorithm for size-based task assignment, which we refer to as SITA-E — Size Interval Task Assignment with Equal Load. The idea is simple: define the size range associated with each host such that the total work (load) directed to each host is the same. The motivation for doing this is that it is easily proven that balancing the load minimizes mean waiting time. By operating a distributed server under unbalanced load at the hosts, the throughput is the same, but the mean waiting time increases.

The mechanism for achieving balanced expected load at the hosts is to use the *task size distribution* to define the cutoff points (defining the ranges) so that the expected work directed to each host is the same. The task size distribution is easy to obtain by maintaining a histogram (in the dispatcher unit) of all task sizes witnessed over a period of time. The cutoff points are computed once, given the distribution, and then the dispatcher unit need only keep a record of these cutoff points for implementing the SITA-E policy. However, one benefit of SITA-E is that if the task size distribution were to change significantly over time, the cutoff points could be recomputed, to adapt to this change.

More precisely, Size Interval Task Assignment with Equal Load (SITA-E) relies on the assumption that the distribution of the size of incoming requests is known, and has finite mean M. Let $F(x) = \Pr\{X \leq x\}$ denote the cumulative distribution function of request sizes and $f(x) = \Pr\{X = x\}$ the corresponding density function. Let k denote the smallest possible request size, p (possibly equal to infinity) denote the largest possible request size, and h be the number of hosts. Then we determine "cutoff points" x_i, $i = 0 \ldots h$ where $k = x_0 < x_1 < x_2 < \ldots < x_{h-1} < x_h = p$, such that

$$\int_{x_0=k}^{x_1} x \cdot dF(x) = \int_{x_1}^{x_2} x \cdot dF(x) = \int_{x_2}^{x_3} x \cdot dF(x) = \cdots = \int_{x_{h-1}}^{x_h=p} x \cdot dF(x) = \frac{M}{h} = \frac{\int_k^p x \cdot dF(x)}{h}$$

and assign to the ith host all files ranging in size from x_{i-1} to x_i.

SITA-E as defined can be applied to *any* task size distribution with finite mean. In the remainder of the paper we will always assume the task size distribution is the Bounded Pareto distribution, $B(k, p, \alpha)$, and we will examine SITA-E's performance in that case.

4 Simulation Results

In this section we compare the following task assignment policies using simulation:

1. Random: an incoming task is sent to host i with probability $1/h$.

2. Round-Robin: tasks are assigned to hosts in cyclical fashion with the ith task being assigned to host i mod h.

3. SITA-E: size-based assignment in which load is balanced in expectation.

4. Dynamic: incoming tasks are assigned to the host with the smallest amount of outstanding work.

As described in Sections 3, tasks arrive according to a Poisson process, and are drawn independently from a $B(k, p, \alpha)$ distribution. We initially consider an eight host system ($h = 8$) operating at a utilization (ρ) of 0.8.

Simulating a server system with heavy-tailed, highly variable service times is difficult because the system approaches steady state very slowly and usually from below [2]. This occurs because the running average of task sizes is typically at the outset well below the true mean; the true mean isn't achieved until enough large tasks arrive. The consequence for a system like our own is that simulation outputs appear more optimistic than they would in steady-state. To make our simulation measurements less sensitive to the startup transient, we run our simulation for 4×10^5 arrivals and then capture data from the next single arrival to the system only. This makes our measurements less sensitive to the behavior of the system during its startup transient than if whole-simulation averages were used. Each data point shown in our plots is the average of 400 independent runs, each of which started from an empty system. We choose 4×10^5 arrivals to be the timescale of interest because it is longer than any of our empirical traces of tasks arriving at high performance computing centers (as described in Section 3).

We consider α values in the range 1.1 (high variability) to 1.9 (lower variability). As described in Section 2.2, α values in the range 1.0 to 1.3 tend to be common in empirical measurements of computing systems.

Figure 4 shows the performance metrics of the system for all four policies, as a function of α (note the logarithmic scale on the y axis). Figure 4(a)(left) shows mean waiting time, (b)(left) shows mean queue length, and (c)(left) shows mean slowdown. Below we simply summarize the results of Figure 4. In the next section, we will use analysis to explain these results.

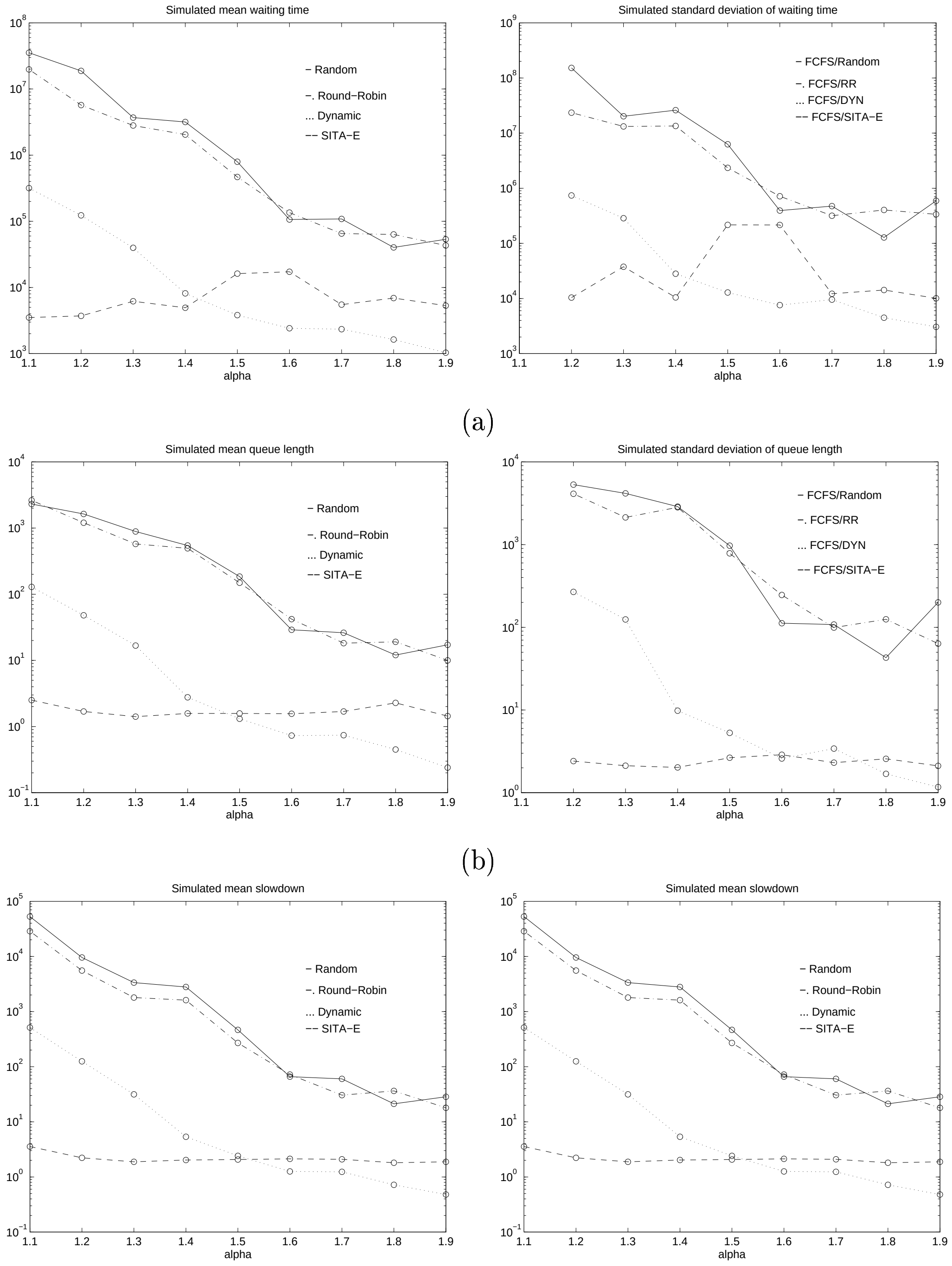
First of all, observe that the performance of the system under the Random and Round Robin policies is similar, and that both cases perform much more poorly than the other two (SITA-E and Dynamic). As α declines, all of the performance metrics under the Random and Round-Robin policies explode approximately exponentially. This gives an indication of the severe impacts that heavy-tailed workloads can have in systems with naive task assignment policies.

The Dynamic policy shows the benefits of instantaneous load balancing. Dynamic is on the order of 100 times better for all metrics when compared to Random and Round Robin. For large α, this means that Dynamic performs quite well—with mean queue lengths and mean slowdown less than 1. However as the variability in task size increases (as α approaches 1), Dynamic is unable to maintain good performance. It too suffers from roughly exponential explosion in performance metrics as α declines.

In contrast, the behavior of SITA-E is quite different from that of the other three. Over the entire range of α values studied, the performance of the system under SITA-E is relatively unchanged. Mean slowdown is always between 2 and 3, and mean queue length is always between 1 and 3. This is the most striking aspect of our data: in a range of α in which performance metrics for Random, Round Robin, and Dynamic all explode, SITA-E's performance remains remarkably insensitive to the increase in task size variability.

As a result we find that when task size is less variable, Dynamic task assignment has better performance; but when task sizes show the variability that is more characteristic of empirical measurements ($\alpha \approx 1.1$), SITA-E's performance can be on the order of 100 times better than that of Dynamic.

While these results represent the behavior of the system at moderate load ($\rho = 0.8$) we have also studied the system at low load ($\rho = 0.5$) and high load ($\rho = 0.9$). Those results exhibit



(a)

(b)

(c)

Figure 4: Performance Metrics under Simulation of Four Task Assignment Policies as a Function of α. Waiting Time (a), Queue Length (b), and Slowdown (c); Left: Mean values; Right: Sample Standard Deviations.

behavior consistent with intuition: at low loads, Dynamic is preferable to SITA-E over a larger range. This follows from the observation that Dynamic is able to explicitly exploit idle hosts, which are more likely at low load. Conversely, SITA-E is preferable to Dynamic over a wider range at high load, because idle hosts are less likely under those conditions.

The remarkable consistency of system performance under the SITA-E policy across the range of α from 1.1 to 1.9 is difficult to understand using the tools of simulation alone. For that reason the next section develops analysis of SITA-E and other policies, and uses that analysis to explain SITA-E's performance.

5 Analysis of Task Assignment Policies

To understand the differences between the performance of the four task assignment policies, we provide a full analysis of the Round-Robin, Random, and SITA-E cases. The Dynamic policy is hard to analyze, however we will prove that it is equivalent to an M/G/h queue, which in turn can be approximated. We will also approximate the performance of the Shortest-Line policy.

In the analysis below we will repeatedly make use Theorem 1 below which analyzes the M/G/1 FCFS queue using the Pollaczek-Kinchin formula:

Theorem 1 *Given an M/G/1 FCFS queue, where the arrival process has rate λ, X denotes the service time distribution, and ρ denotes the utilization ($\rho = \lambda \mathbf{E}\{X\}$). Let W be a task's waiting time in queue, S be its slowdown, and Q be the queue length on its arrival. Then,*

$$\begin{aligned}
\mathbf{E}\{W\} &= \frac{\lambda \mathbf{E}\{X^2\}}{2(1-\rho)} \qquad \textit{[Pollaczek-Kinchin formula]} \\
\mathbf{E}\{S\} &= \mathbf{E}\{W/X\} = \mathbf{E}\{W\} \cdot \mathbf{E}\left\{X^{-1}\right\} \\
\mathbf{E}\{Q\} &= \lambda \mathbf{E}\{W\}
\end{aligned}$$

Proof: The slowdown formulas follow from the fact that W and X are independent for a FCFS queue, and the queue size follows from Little's formula. ■

Observe that every metric for the simple FCFS queue is dependent on $\mathbf{E}\{X^2\}$, the second moment of the service time. Recall that if the workload is heavy-tailed, the second moment of the service time explodes, as shown in Figure 3.

5.1 Analysis of the Policies

Random Task Assignment The Random policy simply performs Bernoulli splitting on the input stream, with the result that each host becomes an independent $M/B(k,p,\alpha)/1$ queue. The utilization at the ith host, is equal to the system utilization (or load), that is, $\rho_i = \rho$. So Theorem 1 applies directly, and all performance metrics are proportional to the second moment of $B(k,p,\alpha)$. Performance is generally poor because the second moment of the $B(k,p,\alpha)$ is high.

Round Robin. The Round Robin policy splits the incoming stream so each host sees an $E_h/B(k,p,\alpha)/1$ queue, with utilization $\rho_i = \rho$. This system has performance close to the Random case since it still sees high variability in service times, which dominates performance.

SITA-E. The SITA-E policy also performs Bernoulli splitting on the arrival stream (which follows from our assumption that task sizes are independent). By the definition of SITA-E, $\rho_i = \rho$. However the task sizes at each queue are determined by the particular values of the interval cutoffs, $\{x_i\}, i = 0, ..., h$. In fact, host i sees a $M/B(x_{i-1}, x_i, \alpha)/1$ queue. The reason for this is that partitioning the Bounded Pareto distribution into contiguous regions and renormalizing each of the resulting regions to unit probability yields a new set of Bounded Pareto distributions. In the Appendix we show how to calculate the set of x_is for the $B(k,p,\alpha)$ distribution, and we present the resulting formulas that provide full analysis of the system under the SITA-E policy for all the performance metrics.

Dynamic The Dynamic policy is not analytically tractable, which is why we performed the simulation study. However, in Theorem 2 below we prove that a distributed system of the type in this paper with h hosts which performs Dynamic task assignment is actually equivalent to an M/G/h queue, for which there exist known approximations.

Theorem 2 *A distributed server with h hosts, as described in this paper, employing the Dynamic task assignment policy is identical in performance to an M/G/h queue.*[3]

Proof: We prove a stronger statement than is claimed in the theorem. We show that if the Dynamic system and the M/G/h system are fed the same arrival sequence of tasks and if ties

[3] Recall, in an M/G/h queue, arriving tasks are placed in one central queue in the order they arrive. When any of the h hosts becomes free, it takes on the task at the head of the central queue.

are arbitrated in the same way in both systems, then the ith task to arrive in both systems is serviced at the same time and at the same queue in both systems. The proof is by induction on i, and is provided in full detail in the Appendix. ■

Fortunately, there exist known approximations for the performance metrics of the M/G/h queue, [12],[16]:

$$\mathbf{E}\left\{Q_{M/G/h}\right\} = \mathbf{E}\left\{Q_{M/M/h}\right\} \cdot \frac{\mathbf{E}\{X^2\}}{\mathbf{E}\{X\}^2},$$

where X denotes the service time distribution, and Q denotes queue length. What's important to observe here is that the mean queue length, and therefore the mean waiting time and mean slowdown, are all proportional to the second moment of the service time distribution, as was the case for the Random and Round-Robin task assignment policies. In fact, the performance metrics are all proportional to the squared coefficient of variation ($C^2 = \frac{\mathbf{E}\{X^2\}}{\mathbf{E}\{X\}^2}$) of the service time distribution.

Other Task Assignment Policies: Shortest-Line Rule We mention the Shortest-Line Rule here simply because it relates to previous work, Section 2. We did not simulate this policy, but it is clear that the performance of this policy is no better than the performance of the M/G/h queue.

Using the above analysis we can compute, or approximate, the performance of the above task assignment policies over a range of α values. The analytically-derived mean waiting time and mean slowdown of the system under these policies is shown in Figures 5 and 6.

Figures 5 and 6 show clearly that the performance of the Random and Dynamic policies grow worse as α decreases which is expected since their performance is directly proportional to the second moment of the service time distribution, shown in Figure 3. In the range $1 < \alpha < 2$, shown in the figures to the right, the mean waiting time and mean slowdown of Random and Dynamic explode. In contrast, in this range ($1 < \alpha < 2$), the mean waiting time and especially mean slowdown under the SITA-E policy is remarkably constant. Since this range of α corresponds to most empirical measurements of process lifetimes and file sizes (see Section 2.2), this is an important range to study. The insensitivity of SITA-E's performance to α in this range is the most striking property of our simulations and analysis, and in Section 5.3 we'll show results from analysis which aid in understanding how this benefit of SITA-E arises.

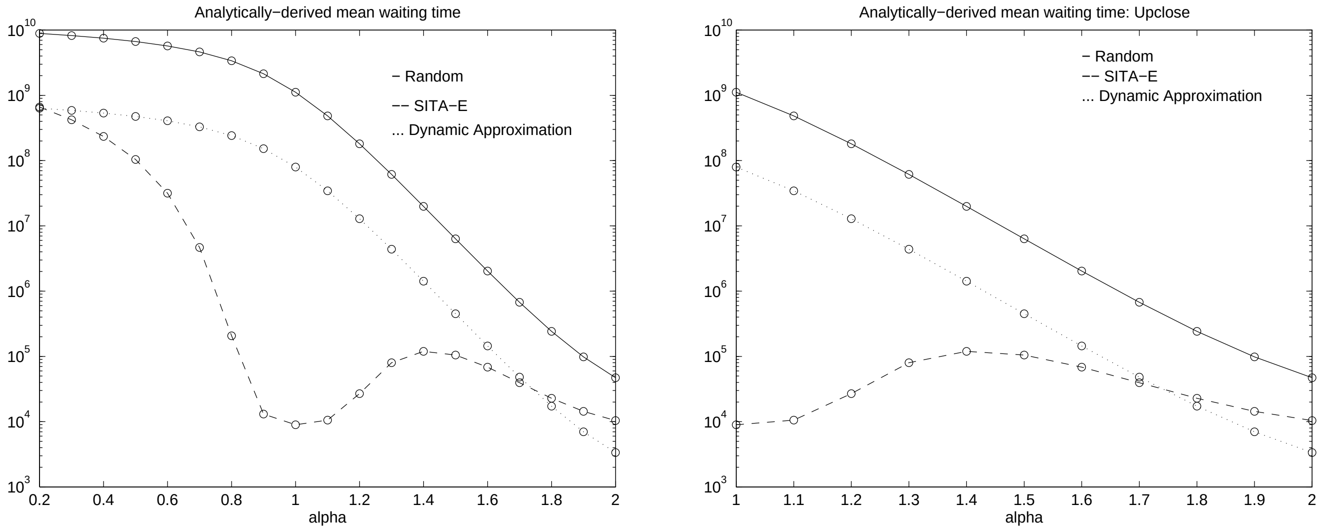


Figure 5: Analysis of Mean Waiting Time. Left: over whole range of α. Right: over range of α corresponding to empirical measurements.

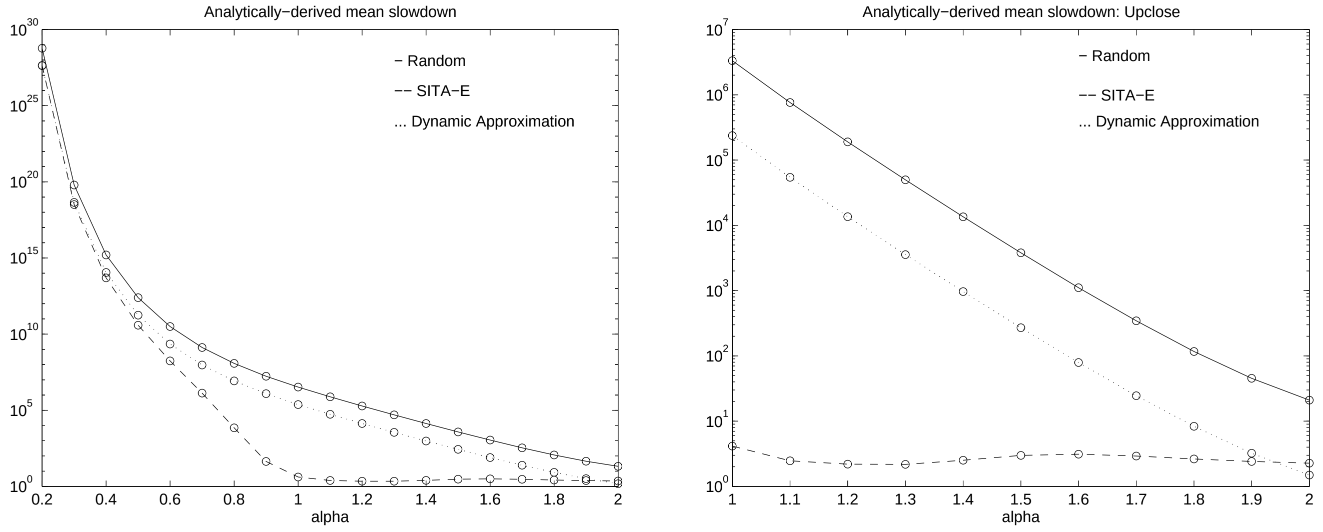


Figure 6: Analysis of Mean Slowdown. Left: over whole range of α. Right: over range of α corresponding to empirical measurements.

5.2 Comparison of Analytic and Simulation Results

The simulation results, Figure 4, are similar in trend to the analytic predictions, yet different enough to warrant discussion. As explained in Section 4, the problem with simulating a highly variable task size distribution is that the mean and variance in the task size distribution witnessed in simulation takes a long time to converge to the true mean and variance, and the process of reaching steady-state occurs from below. Thus the simulations often see a lower mean and especially variance in the task size distribution. Knowing that the variance in the task size distribution is the primary influence on the performance of Random, Round-Robin, and Dynamic, it is understandable that the simulation results for these policies are below the analytically-predicted results. In the case of SITA-E, variability in the task size distribution is less of an issue, as we'll discuss in Section 5.3, so the simulation results are in closer agreement to the analytical results. The combination of these two observations explains why the cross-over point between Dynamic and SITA-E is shifted right in the analysis results, as compared with the simulation results.

5.3 Understanding SITA-E's Performance via Analysis

We now explain SITA-E's good performance, in particular its insensitivity to α in the range $1 < \alpha < 2$.

The benefits from the SITA-E policy derive from three important properties. First, by limiting the range of task sizes at each host, SITA-E drastically reduces the variability of task sizes arriving at most hosts in the system (that is, all but the highest-numbered host). Figure 7 plots the squared coefficient of variation ($C^2 = \sigma^2/\mu^2$) of task sizes arriving at each host as a function of α. For $0 < \alpha < 2$, the C^2 values for all hosts are below the C^2 value for tasks arriving into the system. In fact, for $1 < \alpha < 2$, the C^2 values for hosts 1 through 7 are all less than 1 (the line $C^2 = 1$ is plotted for reference). This indicates that these hosts are seeing very *low* variability in service times, and in fact performing better than an $M/M/1$ queue with utilization ρ.

The second important property of SITA-E is that the majority of tasks are assigned to lower-numbered hosts; as a result the performance at these hosts dominates the mean system performance. This is intensified by the heavy-tailed property, which means that *very* few tasks are assigned to high numbered hosts. Thus as the distribution becomes more heavy-tailed, performance at the low-numbered hosts increasingly determines overall system performance. This effect is shown graphically in Figure 8 which shows the expected waiting time at the ith host ($\mathbf{E}\{W_i\}$) as a function of α, and also shows the mean waiting time for the system ($\mathbf{E}\{W\}$). Note that in the range $1 < \alpha < 2$, although waiting time on host 8 is large, the system is dominated by the performance of hosts 1-7, where waiting time is small. In fact, as α decreases below 1, system performance converges to that of host 1, since in this range of α essentially all tasks are sent to host 1.

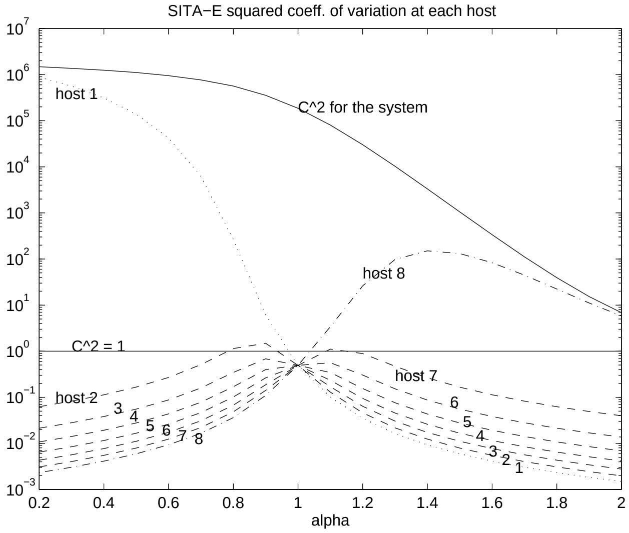


Figure 7: SITA-E values for C^2 on each host (1 through 8), as a function of α.

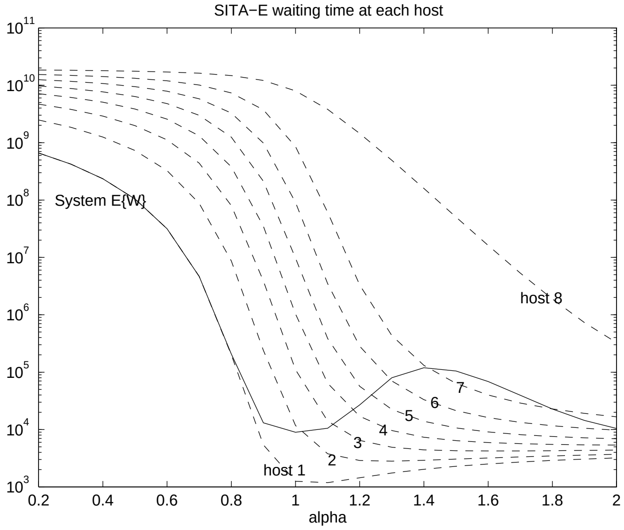


Figure 8: Mean Waiting Time for System and Individual Hosts under SITA-E.

Finally, the third important property of SITA-E is that by partitioning tasks into groups of more nearly equal-sized members, SITA-E results in lower waiting times for smaller tasks, as shown in Figure 8 which depicts far lower waiting times for lower numbered hosts (note the log scale). This has a particularly beneficial effect on slowdown, since the waiting time of a small task is never influenced by service times of large tasks.

The results from analysis also allow us to examine cases in which $\alpha \leq 1$, which are difficult to simulate accurately. In this range of α we see that even under the SITA-E policy, system performance eventually deteriorates badly. To see why, observe that Figure 7 shows that when $\alpha \leq 1$, even host 1 sees extremely high variability in task sizes. The result is that for 8 hosts, SITA-E begins to perform poorly below $\alpha = 1$. Further analysis indicates that adding hosts can extend the range over which SITA-E shows good performance. For example, when the number of hosts is 32, SITA-E's performance does not deteriorate until $\alpha = .8$. That is, for $h = 32$ and $\alpha > .8$, SITA-E has low, nearly-constant slowdown.

Finally, our analysis also helps explain why the Dynamic policy begins to perform poorly in the range $1 < \alpha < 2$. Dynamic attempts to achieve instantaneous load balance and send a task to the queue at which it will wait least. However, our results indicate that this is not the most important goals when variance in task sizes is high. We conclude that when task size variance is high, a more important goal is to reduce the task variance. Since Dynamic does not intrinsically reduce the variability of task sizes assigned to each queue, as α decreases the performance of Dynamic grows steadily worse. More precisely, recall that in Section 5.1 we saw that the approximation of the performance of the M/G/h queue (and therefore, by Theorem 2, of Dynamic) was proportional to C^2 for the system distribution, G. Figure 7 shows the distributional C^2 as a function of α. This is very large compared with the C^2 value witnessed by the lower-numbered hosts under SITA-E.

6 Conclusion

In this paper we study the behavior of a distributed server system under various policies for task assignment. Our focus is on the influence of the variability of the task size distribution in determining which task assignment policy is best. We primarily consider four policies: Random, Round-Robin, SITA-E (a size-based policy), and Dynamic (sending the task to the host with the least-remaining work). Our performance metrics are mean waiting time and mean slowdown, where these are per-task averages.

We find that the best choice of task assignment policy depends critically on the task size distribution, and in particular on the variability of this distribution. We find that when the task sizes are not highly variable, the Dynamic policy is preferable.[4] However, when task sizes show

[4] It is interesting to ask which policy has the best performance when the task size is exponential. We have throughout assumed a Bounded Pareto task size distribution, however the parameter α allows us to affect the variance of the distribution. It turns out that under our particular B(k,p,α) distribution, when $\alpha \approx 2.4$, the

the degree of variability that is more characteristic of empirical measurements (α close to 1), SITA-E is the best performer.

We find furthermore that the magnitude of the difference in performance of these policies can be quite large. The Random and Round-Robin have similar performance and their performance is inferior to both SITA-E and Dynamic by several orders of magnitude. In the range of task size variability that is characteristic of empirical measurements, our results show that SITA-E outperforms Dynamic by close to 2 orders of magnitude.

With respect to the effect of task size variability on each policy, we find that the mean waiting time and mean slowdown of the Random, Round-Robin and Dynamic policies explode as task size variability is increased (α moves from 2 to 1). By contrast, SITA-E's performance is relatively constant in the range $1 < \alpha < 2$ and only begins to deteriorate for some α less than 1. In fact, the mean slowdown under SITA-E in the range $1 < \alpha < 2$ is always between 1 and 3.

These results are particularly relevant because SITA-E is a static policy which means it is easier to implement in practice than a Dynamic-type policy which would require feedback communication between the hosts and dispatcher. Static policies are important in situations where the dispatcher might be located far away from the hosts, *e.g.,* across the Internet.

More important than the above results, though, is the insights about these four policies gleaned from our analysis:

Our analysis of the Random and Round-Robin policies shows that their performance is directly proportional to the second moment of the task size distribution, which explains why their performance deteriorates as the task size variability increases.

To analyze the Dynamic policy, we first prove that it is equivalent to a very different-looking system, which we can approximate. This leads us to the realization that the performance of Dynamic is also (at least approximately) proportional to the second moment of the task size distribution. Thus, although the Dynamic policy comes closes to achieving instantaneous load balance, and furthermore directs each task to the host where it waits the least, those properties are not enough to compensate for the effect of increasing variance in the task size distribution.

In order to understand why size-based policies are so powerful, we introduce the SITA-E policy which is a simple formalization of size-based policies which is defined to force the expected load at each host to be the same. This simple formalization allows us to obtain a full analysis of the

variance of the distribution is the square of the mean (similar to an exponential distribution). In this range of α, the Dynamic policy is the clear winner.

SITA-E policy, which leads us to a 3-fold characterization of its power: (i) By limiting the range of task sizes at each host, SITA-E greatly reduces the variance of the task size distribution witnessed by the host – thereby improving the performance at the host. (ii) When load is balanced, most tasks are sent to low-numbered hosts, which are the hosts with the best performance. (iii) Mean slowdown is improved because small tasks observe proportionately lower waiting times. These 3 properties allow SITA-E to perform very well in a region of task size variability wherein a Dynamic policy explodes.

References

[1] Mark E. Crovella and Azer Bestavros. Self-similarity in World Wide Web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, December 1997.

[2] Mark E. Crovella and Lester Lipsky. Long-lasting transient conditions in simulations with heavy-tailed workloads. In *Proceedings of the 1997 Winter Simulation Conference*, 1997.

[3] 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.

[4] Mor Harchol-Balter and Allen Downey. Exploiting process lifetime distributions for dynamic load balancing. *ACM Transactions on Computer Systems*, 15(3), 1997.

[5] Steven Hotovy, David Schneider, and Timothy O'Donnell. Analysis of the early workload on the cornell theory center ibm sp2. Technical Report 96TR234, Cornell Theory Center, January 1996.

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

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

[8] Randolph D. Nelson and Thomas K. Philips. An approximation to the response time for shortest queue routing. *Performance Evaluation Review*, 7(1):181–189, 1989.

[9] Randolph D. Nelson and Thomas K. Philips. An approximation for the mean response time for shortest queue routing with general interarrival and service times. *Performance Evaluation*, 17:123–139, 1998.

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

[11] David L. Peterson and David B. Adams. Fractal patterns in DASD I/O traffic. In *CMG Proceedings*, December 1996.

[12] S. Sozaki and R. Ross. Approximations in finite capacity multiserver queues with poisson arrivals. *Journal of Applied Probability*, 13:826–834, 1978.

[13] R. W. Weber. On the optimal assignment of customers to parallel servers. *Journal of Applied Probability*, 15:406–413, 1978.

[14] Ward Whitt. Deciding which queue to join: Some counterexamples. *Operations Research*, 34(1):226–244, January 1986.

[15] W. Winston. Optimality of the shortest line discipline. *Journal of Applied Probability*, 14:181–189, 1977.

[16] Ronald W. Wolff. *Stochastic Modeling and the Theory of Queues.* Prentice Hall, 1989.

7 Appendix

This consists of two subsections. The first is the analysis of the SITA-E task assignment policy. The second is the proof of Theorem 2.

7.1 Analysis of SITA-E

The SITA-E task assignment policy can be applied to any task size distribution with finite mean. Below we will provide the full analysis of SITA-E in the specific case where the task size distribution is the Bounded Pareto distribution, $B(k, p, \alpha)$.

We will make use of the fact that if X is a random variable drawn from a $B(k, p, \alpha)$ distribution, then the moments of X are given by:[5]

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

The theorem below shows that there is a simple formula for the x_i, $i = 0 \ldots h$ in the case where the request distribution is $B(k, p, \alpha)$:

Theorem 3 *Let X be a random variable which follows the $B(k, p, \alpha)$ distribution where $0 < \alpha \leq 2$ and k and p are positive constants with $k < p$. Then, if $\alpha \neq 1$, defining*

$$x_i = \left(\frac{(h-i)}{h} k^{1-\alpha} + \frac{i}{h} p^{1-\alpha}\right)^{\frac{1}{1-\alpha}}, \text{ for } i = 0 \ldots h,$$

we have $k = x_0 < x_1 < x_2 < \ldots < x_{h-1} < x_h = p$, and

$$\int_{x_0=k}^{x_1} x \cdot f(x) = \int_{x_1}^{x_2} x \cdot f(x) = \int_{x_2}^{x_3} x \cdot f(x) = \cdots = \int_{x_{h-1}}^{x_h=p} x \cdot f(x) = \frac{\mathbf{E}\{X\}}{h}$$

If $\alpha = 1$, the corresponding definition is:

$$x_i = k\left(\frac{p}{k}\right)^{\frac{i}{h}}, \text{ for } i = 0 \ldots h$$

[5] This formula applies when $\alpha \neq j$; expressions for other cases are given in Theorem 5.

Proof: We demonstrate that when

$$x_i = \left(\frac{(h-i)}{h} k^{1-\alpha} + \frac{i}{h} p^{1-\alpha}\right)^{\frac{1}{1-\alpha}}, \text{ for } i = 0 \ldots h,$$

then the work allocated to the ith host is exactly $\mathbf{E}\{X\}/h$, for all i, assuming $\alpha \neq 1$.

$$\begin{aligned}
\int_{x_{i-1}}^{x_i} x \cdot f(x) dx &= \int_{x_{i-1}}^{x_i} \frac{\alpha k^\alpha}{1-(k/p)^\alpha} x^{-\alpha} dx \\
&= \frac{\alpha k^\alpha}{1-(k/p)^\alpha} \int_{x_{i-1}}^{x_i} x^{-\alpha} dx \\
&= \frac{\alpha k^\alpha}{(1-(k/p)^\alpha)(1-\alpha)} \cdot \left(x_i^{1-\alpha} - x_{i-1}^{1-\alpha}\right) \\
&= \frac{\alpha k^\alpha}{(1-(k/p)^\alpha)(1-\alpha)} \cdot \frac{(-k^{1-\alpha} + p^{1-\alpha})}{h} \\
&= \frac{\mathbf{E}\{X\}}{h}
\end{aligned}$$

If $\alpha = 1$, then defining

$$x_i = k(p/k)^{(i/h)}, \text{ for } i = 1 \ldots h,$$

we have:

$$\begin{aligned}
\int_{x_{i-1}}^{x_i} x \cdot f(x) dx &= \int_{x_{i-1}}^{x_i} \frac{\alpha k^\alpha}{1-(k/p)^\alpha} x^{-\alpha} dx \\
&= \frac{\alpha k^\alpha}{1-(k/p)^\alpha} \int_{x_{i-1}}^{x_i} x^{-\alpha} dx \\
&= \frac{\alpha k^\alpha}{(1-(k/p)^\alpha)(1-\alpha)} \cdot (\ln(x_i) - \ln(x_{i-1})) \\
&= \frac{\alpha k^\alpha}{(1-(k/p)^\alpha)(1-\alpha)} \cdot \left(\frac{1}{h} \cdot (\ln(p) - \ln(k))\right) \\
&= \frac{\mathbf{E}\{X\}}{h}
\end{aligned}$$

■

To analyze the system of h hosts with a SITA-E load balancing policy, we make two observations:

1. The arrival process into each queue is in fact Poisson. To see this, define p_i to be the probability that an arriving task has size between x_{i-1} and x_i. Because the task sizes are uncorrelated, p_i is well-defined. Now, since the arrival process into the system is Poisson with rate λ, it follows that the the arrival process into host i is Poisson with rate λp_i.

2. Each queue may be solved independently as an independent M/G/1 queue where the service time distribution at host i is $B(x_{i-1}, x_i, \alpha)$. That is, partitioning the Bounded Pareto distribution into contiguous regions and renormalizing each of the resulting regions to unit probability yields a new set of Bounded Pareto distributions.

The above discussion is summarized in the theorem below:

Theorem 4 *Assume task arrivals into a system of h hosts with task sizes drawn from the $B(k,p,\alpha)$ distribution. Let X be a random variable from this distribution. Assume the load balancing scheme is SITA-E, where the x_i's are defined according to Theorem 3. Then each queue may be analyzed independently, where the ith queue is an M/G/1 queue with arrival rate λp_i and service times X_i, as defined below.*

$$p_i = \int_{x_{i-1}}^{x_i} f(x)dx = \int_{x_{i-1}}^{x_i} \frac{\alpha k^\alpha x^{-\alpha-1}}{(1-(k/p)^\alpha)}dx = \frac{k^\alpha}{1-(k/p)^\alpha}\left(x_{i-1}^{-\alpha} - x_i^{-\alpha}\right)$$

X_i has a $B(x_{i-1}, x_i, \alpha)$ distribution. The moments of X_i are given by Equation 2, with $k = x_{i-1}$ and $p = x_i$. Furthermore, the load at the ith queue, ρ_i, is equal to the overall load, ρ.

Proof:

To see that X_i has a $B(x_{i-1}, x_i, \alpha)$ distribution, observe that X_i is simply the random variable X, where only values in the range (x_{i-1}, x_i) are permitted. Since X is has a Bounded Pareto distribution, X_i does as well.

To prove that $\rho_i = \rho$ for all i, first observe the following hand-waving argument: In SITA-E the "work" is being split up evenly between the h hosts. Since all the hosts process work at the same rate ($\mu = 1$), this says intuitively that each host is getting the same "load," so all hosts have load ρ. More formally,

$$\begin{aligned} \rho_i &= \lambda_i \mathbf{E}\{X_i\} \\ &= \lambda_i \int_{x_{i-1}}^{x_i} x \frac{f(x)}{p_i} dx \\ &= \lambda_i \cdot \frac{1}{p_i} \int_{x_{i-1}}^{x_i} x f(x) dx \\ &= \lambda \frac{\mathbf{E}\{X\}}{h} \\ &= \rho \end{aligned}$$

where the second equality from the end follows from the definition of SITA-E.

■

The theorem below presents the full analysis of a distributed FCFS server with SITA-E task assignment. As defined in Section 3, the random variables W, S, Q, and X refer to the waiting time, slowdown, queue size and service time, respectively, for the h host system. The variables W_i, S_i, Q_i, and X_i are the corresponding random variables for just the ith queue (ith host) of the system.

Theorem 5 *Given a system of h hosts with a Poisson arrival process with rate λ, task sizes drawn from a Bounded Pareto distribution $B(k,p,\alpha)$, and employing the SITA-E task assignment policy, then:*

$$\begin{aligned} x_i &= \begin{cases} \left(\frac{(h-i)}{h} k^{1-\alpha} + \frac{i}{h} p^{1-\alpha}\right)^{\frac{1}{1-\alpha}} & \textit{if } \alpha \neq 1 \\ k\left(\frac{p}{k}\right)^{\frac{i}{h}} & \textit{if } \alpha = 1 \end{cases} \\ p_i &= \frac{k^\alpha}{1-(k/p)^\alpha}\left(x_{i-1}^{-\alpha} - x_i^{-\alpha}\right) \\ \mathbf{E}\left\{X_i^j\right\} &= \begin{cases} \frac{\alpha x_{i-1}^\alpha (x_{i-1}^{j-\alpha} - x_i^{j-\alpha})}{(\alpha-j)(1-(x_{i-1}/x_i)^\alpha)} & \textit{if } \alpha \neq j \\ \frac{x_{i-1}x_i}{x_i - x_{i-1}}(\ln x_i - \ln x_{i-1}) & \textit{if } \alpha = j = 1 \end{cases} \end{aligned}$$

$$
\begin{aligned}
\mathbf{E}\left\{1/X_i^j\right\} &= \mathbf{E}\left\{X_i^{-j}\right\} \\
\mathbf{E}\left\{W_i\right\} &= \frac{\lambda p_i \mathbf{E}\left\{X_i^2\right\}}{2(1-\lambda p_i \mathbf{E}\left\{X_i\right\})} \\
\mathbf{E}\left\{W\right\} &= \sum_{i=1}^{h} p_i \mathbf{E}\left\{W_i\right\} \\
var(W_i) &= \mathbf{E}\left\{W_i\right\}^2 + \frac{\lambda p_i \mathbf{E}\left\{X_i^3\right\}}{3(1-\lambda p_i \mathbf{E}\left\{X_i\right\})} \\
\mathbf{E}\left\{W_i^2\right\} &= var(W_i) + (\mathbf{E}\left\{W_i\right\})^2 \\
\mathbf{E}\left\{W^2\right\} &= \sum_{i=1}^{h} p_i \mathbf{E}\left\{W_i^2\right\} \\
var(W) &= \mathbf{E}\left\{W^2\right\} - \mathbf{E}\left\{W\right\}^2 \\
\mathbf{E}\left\{Q_i\right\} &= \lambda p_i \mathbf{E}\left\{W_i\right\} \\
\mathbf{E}\left\{Q\right\} &= \sum_{i=1}^{h} p_i \mathbf{E}\left\{Q_i\right\} \\
\mathbf{E}\left\{S_i\right\} &= \mathbf{E}\left\{\frac{W_i}{X_i}\right\} = \mathbf{E}\left\{W_i\right\} \cdot \mathbf{E}\left\{1/X_i\right\} \\
\mathbf{E}\left\{S\right\} &= \sum_{i=1}^{h} p_i \mathbf{E}\left\{S_i\right\} \\
\mathbf{E}\left\{S_i^2\right\} &= \mathbf{E}\left\{W_i^2\right\} \cdot \mathbf{E}\left\{1/X_i^2\right\} \\
\mathbf{E}\left\{S^2\right\} &= \sum_{i=1}^{h} p_i \mathbf{E}\left\{S_i^2\right\} \\
var(S) &= \mathbf{E}\left\{S^2\right\} - \mathbf{E}\left\{S\right\}^2
\end{aligned}
$$

Proof: The h queues are independent and ith queue may be viewed as an M/G/1 queue with arrival rate λp_i and service distribution B(x_{i-1},x_i,α). Since the queues are not identical, evaluating the system performance metrics usually requires a weighted sum over the queues. The important observation for SITA-E is that although W and X are *not* independent for a task entering the system in SITA-E (unlike in the RANDOM task assignment policy), within a single host W and X *are independent*. Thus in the analysis of slowdown performance metrics for the ith host, we can assume W_i and X_i to be independent.

Each formula follows from those above it: The equation for mean and variance of waiting time at the ith queue are given by the Pollaczek-Kinchin formulae. The mean waiting time for the system is the weighted average of the waiting time at the individual hosts. To derive the variance in the system waiting time, we first derive the second moment of the system waiting time by again using a weighted average over the individual hosts. The mean queue size at a host again follows from Little's formula. Since Q is define as the queue size seen by an arrival into the system, the expected value of Q is the weighted average over the queue size at each queue (note, Little's law does not apply to our definition of Q). The slowdown derivations are first done for an individual host, where we know that waiting time, W_i, and service time, X_i, are independent. Then the mean and second moment of the system slowdown are derived by taking a weighted average over those values at the individual hosts. ■

7.2 Proof of Theorem 2

Proof:
We will make use of the following useful observation in our proof:
Useful Observation: In both systems, the ith task to arrive at the system can't begin service before the (i-1)th task begins service.
Assume both systems receive identical arrival sequences of tasks. Assume also ties are resolved in same way in both systems.
Inductive hypothesis: Each task is served by the same host in both systems. The task begins and ends service at the same time in both systems.
Base case: True for first task.
Consider the ith task to arrive to the systems, and assume inductive hypothesis for all tasks arriving before it. We show 3 things:

1. First we show ith task is served by M/G/h no later than by Dynamic:

   Assume that in the Dynamic system the ith task is served by host j at time t. This means that at time t some task, call it q, just completed service at host j in the Dynamic system. By the inductive hypothesis, task q also just completed service, at host j, in the M/G/h system. Thus at time t, the $j$thhost in the M/G/h system is ready for a new task. We show the ith task in M/G/h is either the front of the central queue at time t, or already serving. Suppose not, i.e., suppose the ith task is not yet at the head of the central queue at time t. This implies the $(i-1)$th task is still in the central queue, which means, by the inductive hypothesis, that in the Dynamic system, the $(i-1)$th task hasn't yet started receiving service at time t. But this is impossible since by the above "useful observation," the ith task won't start receiving before the $(i-1)$th task has started receiving service.

2. Now we show that the ith task is served by Dynamic no later than by M/G/h.

   Suppose the ith task is served in M/G/h by queue j at time t. This means that at time t some task, call it q, just completed service at host j in M/G/h. By the inductive hypotheis, task q also just completed service, at host j, in the Dynamic system. We show the ith task in the Dynamic system is either the next to serve in queue j or already serving elsewhere. Suppose not, i.e. suppose task i won't be serving for some time yet in the Dynamic system. Thus some other task, v, is next to serve in queue j in the Dynamic system. Then by our "useful observation," v must be an arrival to the system prior to i. Thus the inductive hypothesis applies, which means that v is served at time t in queue j in the M/G/h system too, which is a contradiction.

3. Now we use the fact that the ith task is served by both systems at the same time to show that in both systems service of the ith task occurs at the same queue.

   Let t be the time at which the ith task is served in both systems. Let service in Dynamic occur at queue j. Assume, for contradiction, that in M/G/h, ith task is starting service at time t at queue j' not equal to j. Thus j' is empty at time t in M/G/h and j is not. So some task q must currently be serving at j in M/G/h, and must have started service before time t. By the "useful observation", q arrived to the system prior to the ith task. But then the induction hypothesis applies to q, so q is also currently serving at queue j in the Dynamic system at time t, which is a contradiction.

■