The Optimal Control of Heterogeneous Queueing Systems: A Paradigm for Load-Sharing and Routing

SCOTT SHENKER AND ABEL WEINRIB

Abstract—**The essence of the basic control decisions implicit in load sharing and routing algorithms is captured in a simple model of heterogeneous queue control. We solve for the optimal control policy and investigate the performance of previously proposed policies in a tractable limit of this model. Using our understanding of this solvable limit, we propose heuristic policies for the general model. Simulation data on these policies suggest that they perform well over a wide range of system parameters.**

Index Terms—**Control of queues, heterogeneous systems, multiserver systems, routing problems, stochastic optimization, stochastic scheduling.**

I. Introduction

DISTRIBUTED computing systems composed of networked workstations and servers are now commonplace. Such systems are rapidly becoming larger, both in geographic extent and also in the number of separate computing elements. Distributed systems routinely contain special purpose machines (such as "cycle servers") and, because of the rapid pace of technological advance, they often include computing elements from several different technology generations. All of these factors make diversity, or heterogeneity, an increasingly important issue in distributed systems. The goal of distributed system design is to effectively harness the collective power of the constituent elements. Often, the underlying algorithms have been designed with the assumption of homogeneity; naively applying these algorithms to a heterogeneous system can result in surprisingly poor performance. It is a relatively unexplored but crucial challenge to design distributed systems that remain efficient in the presence of heterogeneity.

Many of the problems that arise in designing distributed algorithms to operate in the presence of heterogeneity involve controlling queues with servers of differing speeds. For example, load balancing algorithms in heterogeneous distributed computing environments, even with free and instantaneous communication, require nontrivial decisions about when and where to process jobs remotely [2], [5], [28]. Another example is routing in computer networks which often involves selecting one of a number of possible nonequivalent routes [9], [11], [12], [29]. Given a specific optimization criterion, such as minimizing waiting time, one would like to determine the optimal decision policy. If optimal policies are unavailable, then approximately optimal *heuristic* policies are desired. The purpose of this paper is to provide some basic insight leading to such heuristic policies. This work builds on two previous papers [24], [28], and some of the results in Section III were first reported in [24].

Manuscript received March 20, 1989; revised July 21, 1989.

S. Shenker is with the Xerox Palo Alto Research Center, Palo Alto, CA 94304.

A. Weinrib is with Bellcore, Morristown, NJ 07960.

IEEE Log Number 8931169.

Instead of focusing on any specific application, we will study the simple abstract model illustrated in Fig. 1(a) of a single queue feeding multiple servers (for other work on this model, see [1], [14], [15], [18], [19], and [27]). The arrival process is Poisson with strength λ and the servers are exponential with service rates μ_k. We are modeling *server-rate* heterogeneity: in general $\mu_j \neq \mu_k$ for $j \neq k$. A controller for the queue chooses when and where to send jobs to be served. Our goal is to find a decision policy that minimizes the average delay (time spent in service plus time spent in the queue). The controller can decide to send a job to an open server (choosing the fastest open server), or to hold the job in the queue until a faster server becomes available. For homogeneous systems, where all of the servers are equivalent, there is no incentive to retain jobs in the queue. In heterogeneous systems, queueing jobs is often preferable when the only available servers are relatively slow. There is a tradeoff between the extra time spent in the queue waiting for a faster server versus the extra time spent in service at the slower server.

Given values for the system parameters, there are techniques to compute the optimal policy (see [7]). For large systems, these methods are not tractable analytically and rapidly become too cumbersome to implement numerically (since they yield a set of self-consistent equations whose cardinality grows exponentially in the number of servers). Even in the smallest nontrivial case of just two nonidentical servers, solving for the optimal control policy is difficult (but possible, see [15] and [27]). While optimal solutions are desirable, they generally appear beyond our present technical means.

In the absence of optimal solutions, we turn to devising close-to-optimal *heuristic* policies. In Section II, we introduce several previously suggested heuristic control policies, and describe how they apply to our particular model. Each of these heuristics can be expressed as minimizing (or maximizing) some quantity on a job-by-job basis. The choice of the job-by-job optimization principle used in these heuristics is based merely on intuition, and their evaluation has previously been limited to some isolated simulation results.

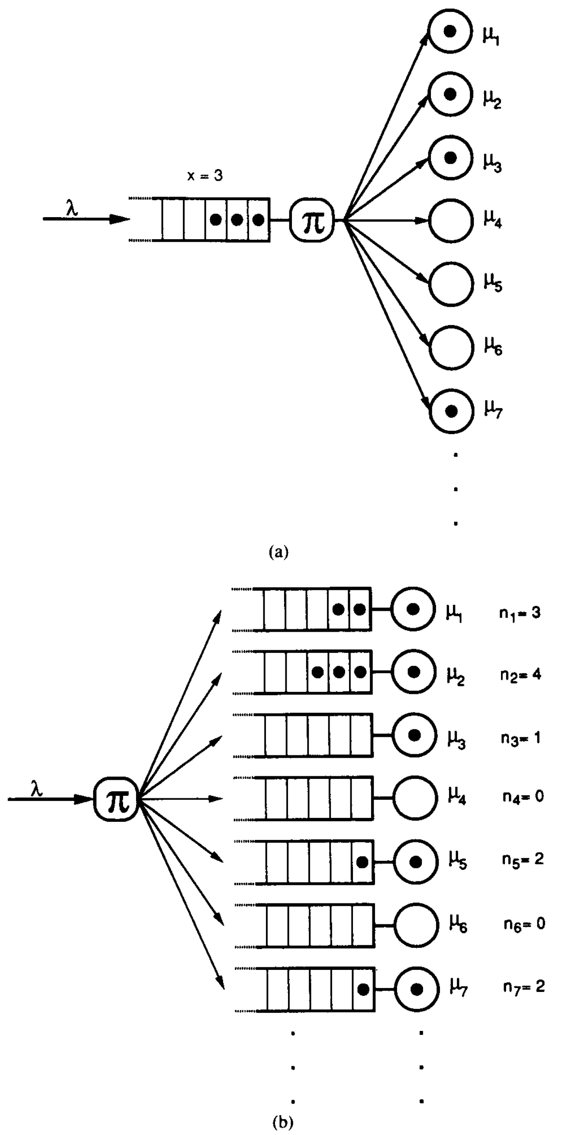


Fig. 1. The heterogeneous queueing models; the scheduler executes policy π to place jobs so as to minimize the average delay. (a) The single queue model with queue length $x = 3$ and servers of rate μ_k. (b) The parallel queue model with queue lengths n_k and servers of rate μ_k.

The first goal of this paper is to provide a more thorough understanding of the performance of these currently available heuristic policies. In Section III, we analyze a simplified and solvable special case of our general queue control model, one in which there are only two server speeds with some number of the fast servers and an infinite number of the slow servers. We explore a variety of limiting extremes (an infinite number of fast servers, a large but finite number of fast servers, and only one fast server with a large ratio in service rates). In each of these regimes, we solve for the optimal policy and compare it to the heuristic policies. This allows us to rigorously identify certain limits in which each of these policies perform significantly suboptimally. In Section IV, we reanalyze this two-speed model using the technique of *policy iteration* (details of the calculation are relegated to the Appendix). We calculate the exact cost in terms of delay to the system of queueing or serving a job, and thus identify the job-by-job optimization principle that produces the optimal policy.

The second goal of this paper, addressed in Section V, is to use our understanding of the simple two-speed system to generate new heuristic policies for the general model in a principled fashion. The key step in this process involves mapping every scheduling decision in the general case to a corresponding one in the two-speed model. The mapping from the general-speed model to the two-speed model is approximate, but the two-speed decision is correct. Thus, our heuristic policy can be viewed as an approximate calculation of the correct job-by-job optimization principle; the other heuristics do an exact calculation of a suboptimal optimization principle.

The final goal of this paper is to demonstrate, through simulation studies, that there are model systems for which the performance of the heuristic policies (ours and others) can be distinguished; simulations of fairly small systems have suggested that the performance of various policies are quite similar (e.g., [19]). We concentrate on fairly large systems, with a large number of slow servers; these systems illustrate the differences between the policies, and are also good models for many current networked systems where hundreds of (slow) workstations are combined with various faster "cycle-servers." In the simulation study discussed in Section VI, we compare the heuristic policies on two different server configurations, finding that our policies outperform all of the previously suggested heuristic policies.

The insight generated in Sections III and IV would be of limited utility if it were only applicable to our specific single queue/multiple server model. In Section VII, we test the robustness of this insight, as well as address another important model of distributed systems, by considering the parallel queue model illustrated in Fig. 1(b). We generate a new heuristic policy for this case and compare it to previously suggested heuristics through simulation.

II. Heuristic Policies

We now define three heuristic queue control policies. Queue control policies are decision rules that, given the set of busy servers and the number of jobs in the queue, determine whether or not to send a job to the fastest open server. Since here we are only concerned with the average delay, and not higher moments, it does not matter which queued job is sent to the open server. However, we will find it useful to say that a controller would send the job in queue position x to the open server if, given a total queue length of x, the queue control decision is to send some job to the server.

A natural candidate for an effective heuristic policy is the *shortest expected delay* (SED) policy, where the controller chooses the option that minimizes the expected delay for each individual job [10], [13]. This policy sends a job in position x in the queue to the fastest open server if

$$x > \frac{\sum_{\mu_k > \mu_{\text{open}}} (\mu_k - \mu_{\text{open}})}{\mu_{\text{open}}} \tag{2.1}$$

where μ_{open} denotes the speed of the open server, and we use the convention that x does not count those jobs presently in service. The condition in (2.1) can be motivated as follows. The expected delay of a job in position x in the queue that is queued for a faster server depends on the scheduling decision of the jobs ahead of it. If we assume that all jobs ahead of position x use only those servers faster than μ_{open}, and that these servers are never left open while there are jobs in the queue, then this expected delay is

$$\frac{x + \sum_{\mu_k > \mu_{\text{open}}} 1}{\sum_{\mu_k > \mu_{\text{open}}} \mu_k} \tag{2.2}$$

where the sum is over all of the servers that are faster than the fastest open server. In contrast, the expected delay of a job that is served immediately by the fastest open server is simply $1/\mu_{\text{open}}$. The condition in (2.1) results if this delay $1/\mu_{\text{open}}$ is compared to the delay of (2.2) for each job in the queue. Note, however, that the delay calculated in (2.2) is not the actual expected delay since the preceding jobs, in following the SED policy themselves, may not keep all of the servers faster than μ_{open} occupied. Nonetheless, and for reasons explained in [13], the policy defined by (2.1) does indeed minimize the expected delay for each job.

Agrawala *et al.* [1] have shown that this SED policy minimizes the average delay of a given batch of jobs with no subsequent arrivals. However, with an ongoing arrival process the SED policy no longer always minimizes the average delay. Scheduling decisions affect not only the job being scheduled but also potentially affect subsequently arriving jobs, which may be forced to wait in the queue longer or to use a slower server as a result of previous scheduling decisions. Nonetheless, one might expect that the heuristic of minimizing the delay for each individual job would at least provide close-to-optimal performance. Simulation data from Nelson and Towsley [19] on relatively small systems suggests that SED performs almost as well as other available heuristics. To the contrary, we will demonstrate that SED is significantly suboptimal for large systems with large heterogeneity in speeds; for these systems, the SED policy completely wastes the lower delay available from the faster servers. The SED policy has also been endorsed as close-to-optimal for systems with multiple parallel queues [2], [6]; in Section VII, we shall see that SED can perform poorly for this model as well.

Another possible heuristic is the *never queue* (NQ) policy, where the controller always sends jobs to the fastest available server. This policy might at first appear ill-advised, in that jobs are scheduled in a manner that increases their individual expected delays; however, the NQ policy minimizes the extra delay caused to the subsequently arriving jobs, so that the effect on the overall average delay may be reasonable. We shall see that NQ is asymptotically optimal for large systems. However, it does not perform well for smaller systems with large heterogeneity.

The SED policy minimizes the expected delay of each arriving job. Similarly, the NQ policy can be seen as maximizing the instantaneous throughput rate, i.e., the total service rate of all occupied servers immediately following a scheduling decision. Maximizing the throughput rate only at the instant of scheduling is shortsighted. An alternative throughput optimization criterion is to maximize the expected number of job completions before the next job arrival. This policy will be called the *greedy throughput* (GT) policy. Nelson and Towsley have introduced such a policy [18], [19] (see also [21]); they present general formulas for its implementation which are too complicated to redisplay here. Chow and Kohler [5] define a somewhat similar policy for the parallel queue model. The GT policy outperforms both NQ and SED for most of the systems we investigate, but is outperformed, in turn, by the policies that we generate in Section V.

The job-by-job optimization principle guiding each of these policies (individual delay, instantaneous throughput, and throughput before next job arrival) have been justified only on the basis of intuition. The resulting policies have been previously evaluated only through limited simulation. In the next section, we calculated their performance in the context of a simple model, giving us a more systematic evaluation of these policies.

III. Simplified Two-Speed Model

Consider a system with only two classes of servers, fast and slow, with m fast servers and an infinite number of slow servers. The restriction to two classes of servers introduces the simplest form of heterogeneity. An infinite number of slow servers is equivalent to a large but finite number of slow servers, except in the heavy traffic regime [20], [24]. With these two simplifications, this model is now equivalent to the problem of admission to an M/M/m queue with m fast servers when one identifies the average time spent in service at the slow servers as the penalty for rejecting a job, and assigns a unit penalty per unit time for waiting in the queue. Much is known about this admission problem (see [17], [25], and [26]). In [16], it is shown that the optimal policy is of a threshold type. A policy with threshold τ will send a job to a slow server whenever there are τ or more jobs ahead of it in the queue (this number does not include those jobs presently in service). Otherwise, jobs are queued up until a fast server becomes available. Our search for the optimal policy now reduces to the search for the optimal threshold τ.

A. Delay and Threshold Formulas

It is helpful to scale the arrival rate by the number of fast se: rs m, defining $\rho = \lambda/m\mu_{\text{fast}}$. The average delay D_τ incurred by a threshold policy with threshold τ can be expressed in terms of quantities calculated for an M/M/m/K queue with $K = m + \tau$:

$$D_\tau = \frac{N_\tau}{m\rho\mu_{\text{fast}}} + \frac{Z_\tau}{\mu_{\text{slow}}} \tag{3.1}$$

where N_τ is the average number of jobs in the M/M/m/K system and Z_τ is the blocking probability (using notation that does not explicitly show the dependence on ρ and m). Both N_τ and Z_τ can be easily expressed in terms of the Erlang-B function, $B(m, m\rho)$:

$$N_\tau = \frac{m\rho + Bm\rho\left(-1 + \frac{1-\rho^\tau}{1-\rho}\right) + B\rho\left(\frac{1-(1+\tau)\rho^\tau + \tau\rho^{\tau+1}}{(1-\rho)^2}\right)}{1 + B\rho\frac{1-\rho^\tau}{1-\rho}} \tag{3.2a}$$

and

$$Z_\tau = \frac{B\rho^\tau}{1 + B\rho\frac{1-\rho^\tau}{1-\rho}}. \tag{3.2b}$$

These formulas permit numerical determination of the optimal τ for a given m and ρ. Let us define the discrete derivative $\Delta(\tau) \equiv D_\tau - D_{\tau-1}$. The full expression for this quantity is complicated, and we merely note that the derivative is proportional to the expression shown below, where we have introduced the notation $R \equiv \mu_{\text{fast}}/\mu_{\text{slow}}$:

$$\Delta(\tau) \propto \tau + m(R-1)(\rho-1) + B\rho\left(m(1-R) + \frac{\tau}{1-\rho} + \frac{\rho^\tau - 1}{(1-\rho)^2}\right). \tag{3.3}$$

The optimal threshold is the largest integral value of τ such that $\Delta(\tau) < 0$. Alternatively, one can compute the real valued solution to $\Delta(y) = 0$ and set $\tau_{\text{opt}} = \lfloor y \rfloor$, where $\lfloor y \rfloor$ denotes the integer part of y. These solutions are monotonically decreasing in ρ and monotonically increasing in R. Furthermore, one can show that the optimal threshold is always between two bounds: $0 \leq \tau_{\text{opt}} \leq m(R-1)$.

The heuristic policies discussed in Section II are threshold policies. Applying formula (2.1) to the two-speed model yields $\tau_{\text{SED}} = \lfloor m(R-1) \rfloor$. The NQ policy has a trivial threshold, $\tau_{\text{NQ}} = 0$. The optimal threshold is always bounded between these two policies. In general, the computation of the GT policy is rather involved [19], [21], but for our simple two-speed model the threshold calculation is straightforward. The increase in expected throughput derived from scheduling an additional job is merely the probability that this job will complete service before the next job arrives. When placing the job at the end of the queue, in position x, this probability is given by $(1+\rho)^{-x}(1+m\rho)^{-1}$. Placing the job on a slow server yields an increase of $(1+Rm\rho)^{-1}$. Thus, the resulting threshold is

$$\tau_{\text{GT}} = \left\lfloor \frac{\ln\left[\frac{1+Rm\rho}{1+m\rho}\right]}{\ln(1+\rho)} \right\rfloor. \tag{3.4}$$

Note that the GT policy is equivalent to the SED policy in the $\rho \to 0$ limit and is equivalent to the NQ policy in the $\rho \to \infty$ limit.

With the threshold expressions in hand, we now turn to evaluating the delay performance of these three policies. We will first analyze the performance in the limit of infinite m, then study the performance for large but finite m, and finally compute the performance in the special case $m = 1$.

B. Infinite System Limit

The behavior of $B(m, m\rho)$ with large m depends crucially on ρ. For $\rho < 1$, $B \to 0$ exponentially fast, so the average delay is just $1/\mu_{\text{fast}}$. The delay is independent of the threshold, so all policies are identical in this limit. For $\rho > 1$, the limiting value of B is $(\rho - 1)/\rho$. The asymptotic delay is then

$$D_\tau = \frac{1}{\mu_{\text{fast}}}\left\{1 + (R-1)\frac{\rho-1}{\rho} + \frac{\tau}{m\rho}\right\}. \tag{3.5}$$

Note that the limiting delay depends only on the limit of τ/m when $m \to \infty$.

Substituting the expression $\tau_{\text{SED}} = \lfloor m(R-1) \rfloor$ into (3.5), the delay for the SED policy with $\rho > 1$ is $D_{\tau_{\text{SED}}} = R/\mu_{\text{fast}} = 1/\mu_{\text{slow}}$. The SED policy makes every server appear as bad as the slow servers, in that the controller keeps adding jobs to the queue until the average delay for a job entering the queue is as long as the expected service time on a slow server. In this regime, SED completely wastes the lower service delay of the fast servers and would perform just as well if given only the slow servers. When m is large, fluctuations in the arrival process are insignificant, so queueing up for the fast servers just creates added delay without any compensating increase in throughput. On the other hand, when m is small, queueing up for the fast servers can increase their throughput by smoothing out the fluctuations.

The minimum $m = \infty$ delay is $D = 1/\mu_{\text{fast}}\{1 + (R-1)(\rho-1)/\rho\}$, and is obtained by any policy that has $\tau/m = 0$ when $m \to \infty$. Note that (3.4) has the limiting form $\tau_{\text{GT}} = \lfloor \ln(R)/\ln(1+\rho) \rfloor$ for $m \to \infty$. Thus, both the NQ and GT policies achieve asymptotically optimal performance in large systems. Fig. 2 shows a graph of the delays for the SED and NQ policies. The discontinuity in the SED delay curve reflects the fact that the SED policy builds up a large queue as soon as it is forced to use any of the slow servers.

We can extend the infinite system results to the more general many-speed case. Consider an arbitrary normalized distribution of server rates $\beta(\mu)$. Let the N-server version of the system consist of N servers with speeds chosen randomly from the distribution $\beta(\mu)$ and an arrival rate of $\lambda_N = N\lambda$. As long as the distribution of server speeds is sufficiently smooth, the limit of an infinite number of servers can be modeled by a fluctuation-free system. The system will utilize just those servers necessary to provide throughput equal to the arrival rate, and the various policies will only affect the number of jobs kept in the queue. In particular,

$$D_{\text{SED}} = \frac{1}{\mu_{\text{open}}} \text{ and } D_{\text{NQ}} = \frac{1}{\mu_{\text{ave}}} \tag{3.6}$$

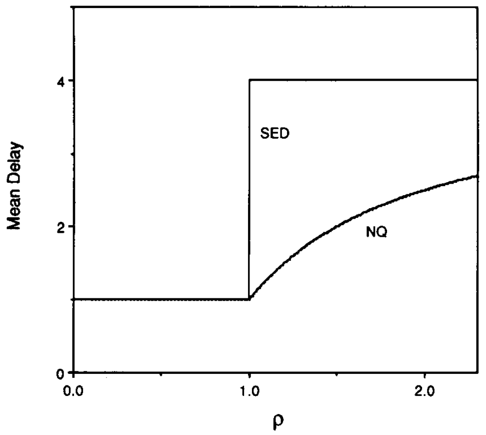


Fig. 2. The $m = \infty$ performance of the NQ and SED policies for the single queue model with servers of two speeds, m of the fast ones, and an infinite number of the slow ones. The scaled arrival rate $\rho = \lambda / m\mu_{\text{fast}}$. $\mu_{\text{fast}} = 1$ and $\mu_{\text{slow}} = 1/4$, so the ratio of speeds is $R = 4$. The delays are calculated according to formula (3.5). Results for large but finite m are very similar to these infinite m results, except that the discontinuity is replaced by a smooth but steeply sloped curve.

where μ_{open} is the solution to

$$\lambda = \int_{\mu_{\text{open}}}^{\infty} d\mu \, \beta(\mu)\mu \tag{3.7}$$

and μ_{ave} is given by

$$\mu_{\text{ave}} = \frac{\lambda}{\displaystyle\int_{\mu_{\text{open}}}^{\infty} d\mu \, \beta(\mu)}. \tag{3.8}$$

Thus, as in the two-speed case, the NQ policy (and any other policy that keeps the queue finite in infinitely large systems), always achieves optimal performance in this infinite limit. The SED policy has performance equivalent to a system where all of the servers have service delay $1/\mu_{\text{open}}$. Fig. 3(a) and (b) depicts the performance of the SED and NQ policies for two distributions of server speeds. Fig. 3(a) is for a discrete distribution, while Fig. 3(b) is for a continuous distribution. In both cases, the NQ policy exhibits markedly lower delay compared to the SED policy.

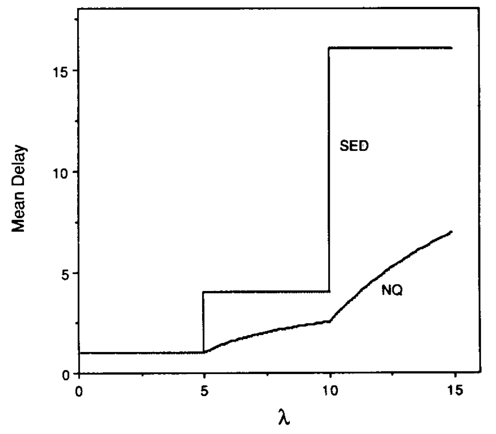
The above results indicate that the SED policy may be significantly suboptimal for infinitely large systems. In this regime, bounded threshold policies like NQ and GT are optimal. In the next section, we examine the behavior for large but finite m. By studying the rate of convergence to optimality, we will see that SED is better than NQ for $\rho < 1$.

C. Large m

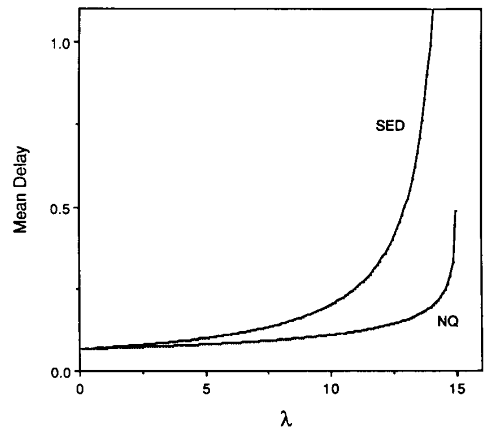
The properties of formula (3.3) for large m depend on the asymptotic nature of $B(m, m\rho)$ (see [8]), and there are three cases that we discuss below.

1) $\rho < 1$: Here, $B \approx (\rho e^{(1-\rho)})^m / m^{1/2}$ so that $\tau_{\text{opt}}(m) = m(R-1)(1-\rho)$ for large m. The delay is given by

$$D_{\tau_{\text{opt}}} = \frac{1}{\mu_{\text{fast}}} + \frac{B}{\mu_{\text{fast}}} \left\{ \frac{1 - \rho^{m(1-\rho)(R-1)}}{m(1-\rho)^2} \right\}. \tag{3.9}$$



(a)



(b)

Fig. 3. The infinite system ($N = \infty$) performance of the NQ and SED policies. (a) The mean delay of a job for a system with a distribution of server speeds given by $\beta(\mu) = (1/21)\delta(\mu - 1) + (4/21)\delta(\mu - 1/4) + (16/21)\delta(\mu - 1/16)$ where the δ denotes the delta function. This is a discrete distribution with only three classes of servers, with $(1/21)N$ fast servers having a service rate of 1, $(4/21)N$ medium servers having a service rate of 1/4, and $(16/21)N$ slow servers having a service rate of 1/16. The weights and speeds were chosen so that each class has the same total processing power. (b) A system with a continuous distribution of server speeds, $\beta(\mu) = 1/\mu$ for $0 \leq \mu \leq 15$, chosen so that each class of server contributes the same total processing power. For both distributions, NQ produces significantly lower delays than SED.

The asymptotic deviations from optimality for the SED, NQ, and GT delays when $\rho < 1$ are

$$D_{\tau_{\text{SED}}} - D_{\tau_{\text{opt}}} \approx \frac{B}{\mu_{\text{fast}}} \frac{\rho^{m(1-\rho)(R-1)}}{m(1-\rho)^2} \tag{3.10a}$$

$$D_{\tau_{\text{NQ}}} - D_{\tau_{\text{opt}}} \approx \frac{B}{\mu_{\text{fast}}} (R-1) \tag{3.10b}$$

$$D_{\tau_{\text{GT}}} - D_{\tau_{\text{opt}}} \approx \frac{B}{\mu_{\text{fast}}} (R-1) R^{\frac{\ln(\rho)}{\ln(1+\rho)}}. \tag{3.10c}$$

SED has a faster exponential convergence rate than NQ or GT. Even though all of these policies are equivalent to the optimal policy in the limit of infinite m, the rate of convergence to optimality can become very slow. For $\rho = 1 - \epsilon$ for small ϵ, the deviation of NQ from optimality goes as $e^{-m\epsilon^2/2}/m^{1/2}$. For $m < \epsilon^{-2}$, the convergence rate will be dominated by the square root term.

2) $\rho = 1$: Here, $B \approx (2/\pi m)^{1/2} - 4/3\pi m + O(m^{-3/2})$. For large m, $\tau_{\text{opt}} \propto m^{1/2}$ and $D_{\tau_{\text{opt}}} - 1/\mu_{\text{fast}} \propto m^{-1/2}$. The asymptotic delay of the SED policy is $D_{\tau_{\text{SED}}} = (R+1)/2\mu_{\text{fast}}$, which is not asymptotically optimal. The NQ and GT policies are asymptotically optimal, both having delays whose deviations from the optimal delays decrease as $m^{-1/2}$.

3) $\rho > 1$: Here, $B \approx (\rho - 1)/\rho + 1/m\rho(\rho - 1) + O(m^{-2})$. For large m, the optimal τ is given by

$$\tau_{\text{opt}} = \left\lfloor \frac{\ln (R)}{\ln (\rho)} \right\rfloor. \tag{3.11}$$

The deviation from optimality for the NQ and GT policies decreases as m^{-1} while the SED policy has deviations that remain finite in the limit: $D_{\tau_{\text{SED}}} \approx 1/\mu_{\text{slow}}$.

While the NQ and GT policies achieve optimal performance in the limit of infinite m, the rate of convergence to this limit depends on ρ. For $\rho < 1$, the convergence is exponentially fast, while for $\rho > 1$ it converges as m^{-1}. For $\rho = 1$ the convergence slows to $m^{-1/2}$. Thus, even for large systems, it might be advantageous to use a policy more sophisticated that NQ or GT in the crossover region of $\rho \approx 1$, where one first needs to utilize the slow servers. In the next section, we investigate the single fast server $m = 1$ case, which reveals precisely how far from optimal these policies are in the crossover region.

D. $m = 1$

We can further explore the performance of the heuristic policies in the limit of large R for the case $m = 1$. The expression (3.1) for the average delay becomes, with $m = 1$,

$$D_\tau = \frac{(1 - (\tau + 2)\rho^{\tau+1} + (\tau + 1)\rho^{\tau+2} + R(1 - \rho)^2\rho^{\tau+1})}{\mu_{\text{fast}}(1 - \rho)(1 - \rho^{\tau+2})}. \tag{3.12}$$

The NQ policy yields a delay that always grows linearly with R: $D_{\tau_{\text{NQ}}} = (1 + R\rho)/\mu_{\text{fast}}(1 + \rho)$. When $\rho < 1$, the optimal threshold grows linearly with R, $\tau_{\text{opt}} \approx R(1 - \rho)$, and the optimal delays as well as the SED delay converge to the M/M/1 result of $D \approx 1/\mu_{\text{fast}}(1 - \rho)$. Thus, the NQ policy is far from optimal for large R when $m = 1$ and $\rho < 1$. The behavior of the GT policy is somewhat peculiar. Define $\rho_c = (\sqrt{5} - 1)/2 \approx 0.618$. For $\rho < \rho_c$, the GT delay converges to the optimal M/M/1 result. When $1 > \rho > \rho_c$, the delay increases as $R^{1+\ln(\rho)/\ln(1+\rho)}$. Apparently in this regime the GT policy does not queue up sufficiently to prevent the delay from diverging.

For $\rho > 1$, $D_{\tau_{\text{SED}}} \approx R/\mu_{\text{fast}}$ and $D_{\tau_{\text{GT}}} \approx R(\rho - 1)/\mu_{\text{fast}}\rho$. For comparison, $\tau_{\text{opt}} \approx \ln(R)/\ln(\rho)$ and $D_{\tau_{\text{opt}}} \approx R(\rho - 1)/\mu_{\text{fast}}\rho$. While all four delay expressions increase linearly with R, the SED and NQ delays have a larger slope than the optimal threshold and GT delays.

So far, for each limiting case we have considered, at least one of the available heuristic policies has been asymptotically optimal. This is no longer true when we consider the intermediate case of $\rho = 1$. With $\rho = 1$,

$$D_{\tau_{\text{GT}}} = \frac{1}{\mu_{\text{fast}}} \left[\frac{R}{1 + \frac{\ln(1+R)}{\ln(2)}} + \frac{\ln(1+R)}{2\ln(2)} \right] \tag{3.13a}$$

$$D_{\tau_{\text{SED}}} = \frac{1}{\mu_{\text{fast}}} \left[\frac{R}{2} + \frac{R}{R+1} \right] \tag{3.13b}$$

$$D_{\tau_{\text{opt}}} = \frac{1}{\mu_{\text{fast}}} \left(\frac{R}{1/2 + (2R + 1/4)^{1/2}} - 1/2(1/2 - (2R + 1/4)^{1/2}) \right). \tag{3.13c}$$

Thus, for $\rho = 1$ and $m = 1$, while both the NQ and SED delays increase linearly with R, and the GT delay increases as $R/\ln(R)$, the optimal threshold policy yields a delay that increases only as $R^{1/2}$.

The introduction of the simple two-speed model, and the subsequent analysis of the three extreme cases, that of infinite systems, large but finite systems, and small systems with extreme heterogeneity, have illuminated the performance characteristics of the various heuristic policies. The bound $\tau_{\text{SED}} \geq \tau_{\text{opt}}$ implies that the SED policy always overqueues for fast servers, causing significant deviation from optimality for large systems at high loads. The NQ policy always underqueues, and while NQ is asymptotically optimal for large systems, it performs poorly in lightly loaded small systems with a large degree of heterogeneity. In contrast to the NQ and SED policies, the GT policy is explicitly dependent on the load λ. Adjusting to the load allows this policy to be asymptotically optimal for large systems and also for both heavily and lightly loaded small systems. However, its performance is significantly suboptimal in, and slightly below, the delicate crossover region $\rho \approx 1$ for small systems with large heterogeneity.

IV. Optimal Cost Function

We have seen that the SED, GT, and NQ policies fail to perform optimally on our simple two-speed model. Before devising better policies, we want to understand why these policies fail. The objective function we are trying to minimize is the average delay experienced by all jobs entering the system; the policy that achieves this objective is sometimes referred to as the *socially* optimal policy [3]. In contrast, the SED policy is an *individually* optimal policy, since it is equivalent to having each job minimize its own individual delay without any consideration of the additional delay experienced by subsequent jobs. The fact that *individually* optimal queue control policies are not necessarily *socially* optimal was first discussed by Naor [17].

Define a job's *social delay* to be the sum of both its individual delay and also the delay its presence causes other jobs. In this section, we exactly calculate this social delay for our two-speed model. The optimal job-by-job optimization principle can then be stated as minimizing the social delay for each job. None of the heuristic policies of Section II calculate this social delay, which is what renders them suboptimal on our simple two-speed model.

For our two-speed model, we define two social delays. The first, $C_\tau^{\text{fast}}(x)$, is the cost in delay of queueing a job at position x in the queue for a fast server when the control policy being employed is a threshold policy with threshold τ. This delay is to be compared to the cost of immediately serving the job at a slow server, which is merely the processing time at the slow server and is independent of the threshold τ: $C^{\text{slow}} = 1/\mu_{\text{slow}}$. To calculate $C_\tau^{\text{fast}}(x)$, consider a system that at time 0 has a queue of length $x - 1$, and place a single test job at the end of the queue. We want to compute the total additional delay experienced by the system. Define $T_\tau(x, t)$ to be the total delay accumulated up to time t. The eventual rate at which delay accumulates is λD_τ, so for large t we can express the accumulated delay in the form $T_\tau(x, t) \sim V_\tau(x) + t\lambda D_\tau$. The term $V_\tau(x)$ reflects the influence of the starting position, and $C_\tau^{\text{fast}}(x) \equiv V_\tau(x) - V_\tau(x-1)$ is the cost, in terms of total delay, of adding a job onto a queue of length $x-1$. The actual computation of $C_\tau^{\text{fast}}(x)$ is relegated to the Appendix; we use the results of that computation below.

The functional form of $C_\tau^{\text{fast}}(x)$ depends on the value of x relative to the threshold τ. For $x \geq \tau$, we have the result

$$C_\tau^{\text{fast}}(x) = \frac{x+m}{m\mu_{\text{fast}}} + \rho\left(\frac{1}{\mu_{\text{slow}}} - D_\tau\right). \tag{4.1}$$

This expression for the cost function has an intuitive interpretation. The first term represents the individual delay of the job; the second term represents the additional delay caused to other subsequently arriving jobs. This extra delay $\rho(1/\mu_{\text{slow}} - D_\tau)$ is merely the expected number of extra jobs sent to the slow servers times the excess delay of those jobs, i.e., the delay $1/\mu_{\text{slow}}$ minus the average delay D_τ.

For $x < \tau$, we find a recursion relation

$$C_\tau^{\text{fast}}(x) = \frac{x+m}{m\mu_{\text{fast}}} + \rho(C_\tau^{\text{fast}}(x+1) - D_\tau). \tag{4.2}$$

This expression has the same intuitive interpretation as before, except that now the excess delay $C_\tau^{\text{fast}}(x+1) - D_\tau$ is the delay of accepting a subsequent job into the queue minus the average delay D_τ. We can find a direct expression for $C_\tau^{\text{fast}}(x)$ by applying (4.2) to the results from (4.1) for $x = \tau$. We find that for $x \leq \tau$, the cost function can be expressed in the following form

$$C_\tau^{\text{fast}}(x) = \frac{1}{\mu_{\text{slow}}} + \frac{D_\tau - D_{x-1}}{\mu_{\text{fast}} Z_{x-1}}. \tag{4.3}$$

How do these cost functions determine an optimal policy? Howard, in [7], outlines a procedure to find optimal stochastic control policies, called *policy iteration*. We will not present this process in general, but will describe only how it applies to our application here. Policy iteration is a two-step process. The first step is to evaluate the cost function $C_\tau^{\text{fast}}(x)$ for a given policy (which here is characterized by a threshold τ). The second stage of the process is to create a new policy in which, for every x, the policy chooses the option that minimizes the cost functions calculated in the previous step. This new policy is again a threshold policy, where the new threshold τ' is the largest integer that satisfies $C_\tau^{\text{fast}}(\tau') \leq C^{\text{slow}}$. Based on general arguments in [7], this process is guaranteed to produce a new threshold τ' that gives a lower average delay: $D_\tau \geq D'_\tau$. The process is iterated until termination, resulting in the optimal threshold τ_{opt}, which is the largest integer satisfying $C_{\tau_{\text{opt}}}^{\text{fast}}(\tau_{\text{opt}}) \leq C^{\text{slow}}$. Note that this cost-based formulation of the optimality criterion is equivalent to the directly computed optimality condition of Section III.

The canonical policy iteration process uses these cost functions and iterates until termination. However, in this iteration procedure one can equally well use a modified cost function for all x,

$$\hat{C}_\tau^{\text{fast}}(x) \equiv \frac{x+m}{m\mu_{\text{fast}}} + \rho\left(\frac{1}{\mu_{\text{slow}}} - D_\tau\right). \tag{4.4}$$

Note that this is the form of the cost function for $x \geq \tau$ merely extended over the whole range of x. Since both the correct and modified cost functions satisfy the same optimality condition, iteration of this modified cost function will result in the same optimal threshold.

We now rewrite (4.4) in terms of locally measurable averages; this new expression will be used in the next section as the foundation for an adaptive policy that realizes the steps of policy iteration dynamically. Define $\bar{x}_\tau$ as the average value of the queue length, and $\bar{i}_\tau$ as the average fraction of idle time of the fast servers. Then, the average occupancy N_τ takes the form $\bar{x}_\tau + m(1 - \bar{i}_\tau)$. Also, the total throughput of the fast servers is just $m\mu_{\text{fast}}(1 - \bar{i}_\tau)$, so the blocking rate Z_τ can be expressed as $Z_\tau = 1 - m\mu_{\text{fast}}(1 - \bar{i}_\tau)/\lambda$. The delay D_τ becomes

$$D_\tau = \frac{\bar{x}_\tau + m(1 - \bar{i}_\tau)}{\lambda} + \left[1 - \frac{m\mu_{\text{fast}}}{\lambda}(1 - \bar{i}_\tau)\right]\frac{1}{\mu_{\text{slow}}}. \tag{4.5}$$

Using this expression in (4.4) for the modified cost function,

$$\hat{C}_\tau^{\text{fast}}(x) = \frac{1}{m\mu_{\text{fast}}} \cdot \left\{(x - \bar{x}_\tau) + m\mu_{\text{fast}}(1 - \bar{i}_\tau)\frac{1}{\mu_{\text{slow}}} + m\bar{i}_\tau\right\}. \tag{4.6}$$

We now have a cost function that is expressed solely in terms of system averages $\bar{x}_\tau$ and $\bar{i}_\tau$, the instantaneous system parameter x, and the system configuration (i.e., m, μ_{fast}, and μ_{slow}).

V. New Policies

The heuristic policies introduced in Section II are based on job-by-job optimization principles that lend themselves to exact calculation but are not guaranteed to produce optimal behavior for any heterogeneous system. The two-speed model introduced in Section III allows us to understand the optimal policy of minimizing the social delay C on a job-by-job basis. We cannot calculate the social delay in general-speed models, but we can approximate it by mapping each general-speed scheduling decision to an analogous two-speed scheduling decision. While the mapping is approximate in systems with more than two speeds, the underlying job-by-job optimization principle is rigorously correct. We now turn to the problem of approximating the general-speed model by the two-speed model.

Note that there are nontrivial scheduling decisions to be made only if there is at least one job in the queue and at least one available server. Let the speed of the fastest available server, μ_{open}, play the role of μ_{slow} in the two-speed case; let the average speed of the servers faster than the fastest open server, call it μ_{ave}, play the role of μ_{fast}; and let the number of such servers, call it N_{faster}, play the role of m. This transcription of the general problem to the two-speed problem is basic to the generation of our heuristic policies.

To turn this transcription into an actual policy, we can then follow two different courses. The first policy, the *deterministic* (D) policy, is found by just inserting the variables μ_{open}, μ_{ave}, and N_{faster} into their proper places μ_{slow}, μ_{fast}, and m in formulas (4.1) and (4.3), and then minimizing the social cost for each job. This is exactly equivalent to using formula (3.3) and directly solving for the optimal threshold. The D policy determines a threshold for each server; i.e., whenever the kth server is the fastest open server, the controller follows a threshold policy with threshold τ_k. Note that the SED and GT policies are also threshold policies since they depend only on the length of the queue and the identity of the fastest open server. A nonthreshold policy would also depend on which slower servers were occupied.

The D policy requires detailed knowledge of the arrival rate, and its "derivation" depends crucially on the memoryless nature of the arrival and service processes. In real applications, servers are not typically exponential nor are arrival processes Poisson. A policy that is perhaps more resilient to these deviations from the ideal is an *adaptive* (A) policy. This policy is generated from the form (4.6) for the cost function which depends only on locally measured quantities. By measuring windowed averages, such a policy can settle down to the optimal policy for the two-speed model without *a priori* knowledge of the arrival rate λ. For each server k, we maintain the measured averages for $\bar{i}_k$, the average idle time, and $\bar{x}_k$, the average queue length measured when server k is idle. Then, using the same transcription from the general-speed to two-speed as above, we can rewrite the cost function (4.6) for the general-speed case

$$\hat{C}^{\text{fast}}_{\text{open}}(x) = \frac{1}{\sum\limits_{\mu_k > \mu_{\text{open}}} \mu_k} \left\{ (x - \bar{x}_{\text{open}}) + \frac{1}{\mu_{\text{open}}} \sum_{\mu_k > \mu_{\text{open}}} \mu_k (1 - \bar{i}_k) + \sum_{\mu_k > \mu_{\text{open}}} \bar{i}_k \right\}. \quad (5.1)$$

The A policy uses this formula, along with $\hat{C}^{\text{slow}}(x) = 1/\mu_{\text{open}}$, and always chooses the action with the minimal cost. The measured statistics, $\bar{i}_k$ and $\bar{x}_k$, will define a policy that, in turn, will determine the measured $\bar{i}_k$ and $\bar{x}_k$, and so on. By using running averages for the statistics, one should arrive at a self-consistent solution analogously to the process of policy iteration. We first suggested *adaptive* policies of this general form in [24] and [28]. Bonomi and Kumar [4], and Krishnan [12] have also suggested adaptive policies, with Krishnan making the connection to policy iteration explicit.

VI. Simulations

The simulation study of the various policies was carried out with a simulation package written for this purpose. Each point in the graphs comes from a run of 600 000 jobs, with the first 100 000 jobs discarded to remove any startup transients. Data on the mean job delay are taken in 25 blocks of 20 000 jobs each; from the block statistics the standard error can be estimated, and is always less than 1 percent of the measured delay. Thus, the differences between policies seen on the graphs are large compared to the errors of the simulation. The statistics $\bar{i}_k$ and $\bar{x}_k$ for the A policy are measured using exponentially decaying window averages with a decay constant of 500 job interarrival times.

To test the various policies, we present results on two basic system configurations. The first configuration models the situation of relatively few classes of service rates with many servers in each class; it consists of five fast servers with service rate 16, 20 medium servers of speed 4, and an infinite number of slow servers with service rate 1. Fig. 4(a) depicts the performance of the various policies on this system. Note that outside of the very lightly loaded regime, where one need only use the fast servers, the SED policy performs quite poorly. The NQ policy performs significantly better than the SED policy, except at light loads. The D policy and the A policy, which have virtually identical performance, match SED for light loads and achieve significantly lower delays than NQ throughout the entire range of λ. The GT policy does significantly better than NQ but not quite as well as A or D. The difference between the A and D delay and the GT delay is largest just before the two crossover points $\lambda \approx 80$ and $\lambda \approx 160$.

One would like to compare these policies to a lower bound. The average delay must be greater than the minimum average service time. This minimum average service time, which is the same as the delay for the infinite system NQ policy, is plotted on Fig. 4(a). The lower bound results are significantly below our best policy. To determine if this difference reflects a poor bound or a poor policy, we searched, via simulation, for the optimal set of thresholds τ_{medium} and τ_{slow} ($\tau_{\text{fast}} = 0$ since one would always send a job to an idle fast server). We find that the D policy is within 1/2 percent of the results from the optimal thresholds, which is within the error bars of the simulations. Note that the truly optimal policy may not be a threshold policy (only for the two-speed model is it known that the optimal policy takes this form).

The second system configuration reflects a more heterogeneous situation, consisting of one server at each of the rates of 128, 64, 32, 16, 8, 4, 2, as well as an infinite number of slow servers of rate 1. Fig. 4(b) exhibits the performance of the various policies for this system. The A and D policies again outperform all the other policies throughout the entire

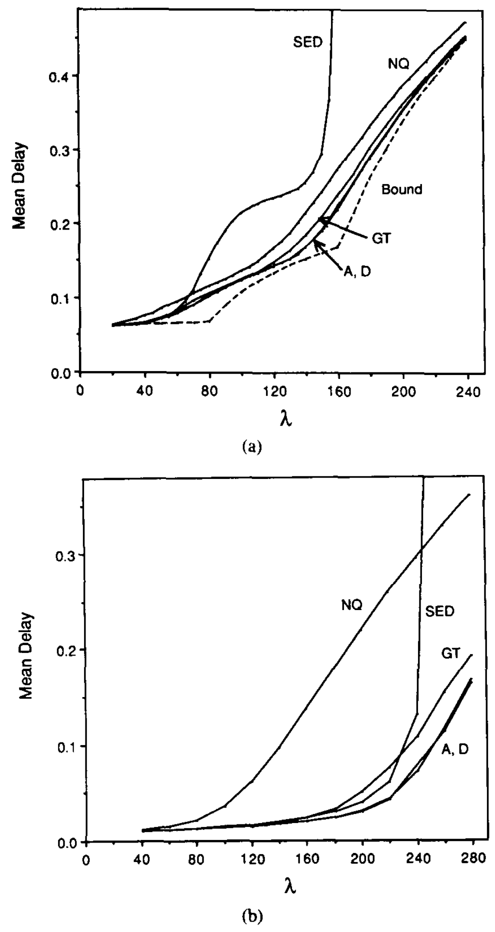


Fig. 4. The performance of the SED, NQ, GT, A, and D policies for the single queue model. The job arrival rate is λ. The curves are derived from a computer simulation of the system. (a) A system with three classes of servers: $\mu_{\text{fast}} = 16$ $m_{\text{fast}} = 5$, $\mu_{\text{med}} = 4$ $m_{\text{med}} = 20$, and $\mu_{\text{slow}} = 1$ $m_{\text{slow}} = \infty$. (b) A system with servers of speeds 128, 64, 32, 16, 8, 4, 2, and an infinite number of servers of speed 1.

parameter range. In this extremely heterogeneous model, the NQ policy performs poorly everywhere. The SED policy performs quite well for $\lambda < 200$, but degrades rapidly above that point. The difference between the GT and the A and D policies is more pronounced here than it was in the first system configuration.

We investigated the robustness of the D policy in three ways. First, since not all systems have exponential servers and Poisson arrivals, we ran the same policies on systems with various combinations of bursty arrivals (Batched-Poisson) and nonexponential servers (both deterministic and bimodal service distributions were tried). While the delay values vary considerably, the relative ranking of the policies remain essentially unchanged, with the A and D policies outperforming all of the other policies. Surprisingly, the D policy, which has none of the adaptive features of the A policy, performs slightly better than the A policy.

Second, we investigated the effect of errors in measurement of the arrival rate λ. The computation of the D policy explicitly uses λ; since it is possible that the arrival rate will be known only approximately, we would like the D policy to be relatively insensitive to errors in λ. Simulations indicate that this is indeed the case. Inserting into the D policy a value for λ that differs by 10 percent from the true arrival rate results in an increase in the system delay of less than 1 percent for most values of λ. The first system displays a maximal increase of 5 percent in the crossover regime $\lambda \approx 160$, and the second system has a maximal increase of 11 percent at high loads.

Third, we tested the effect of finite system sizes. Recall that the D policy was motivated by comparison to systems with an infinite number of slow servers. We performed simulations of two systems identical to those studied above, except that they had only a finite number of the slowest servers; $m_{\text{slow}} = 80$ in the first case and $m_{\text{slow}} = 1$ in the second. In both of these systems, the D policy continues to display lower delays than all of the other policies, indicating that the D policy also performs well for finite systems.

In the examples simulated, the heuristic A and D policies outperform all of the other heuristic policies that we are aware of. Furthermore, for the first system where we obtained the optimal thresholds through exhaustive search, the A and D policies exhibit close to optimal performance. This validates the central approach in Section V of modeling the general-speed model by the simplified two-speed model, and then using the two-speed optimal decision rule. We expected the *adaptive A* policy to be more resilient against nonideal arrival and service processes. Instead, the deterministic D policy actually outperforms, albeit only slightly, the A policy on these nonideal systems. Consequently, we will not consider adaptive policies in the next section.

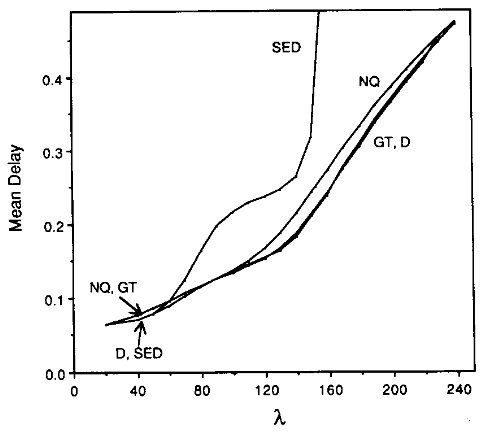
VII. Parallel Queues

The single queue problem considered above reflects those situations where scheduling decisions can be postponed until the job is at the head of a central queue. However, there are many systems where the scheduling decisions must be made immediately and irrevocably upon the job's arrival. These systems are more accurately modeled by the many-queue model of Fig. 1(b). Each server has its own queue, and the scheduler routes each arriving job to the queue selected by the policy. We would like to evaluate several heuristic policies for this system.

A. Policies

There are parallel queue versions of the SED, NQ, and GT queue control policies. Let n_i denote the total number of jobs in the ith queue, including any jobs in service. The SED policy selects the queue that has the minimal expected delay $(n_i + 1)/\mu_i$. The NQ policy chooses the fastest server that has an empty queue; if there are no empty queues, the queue with minimal n_i/μ_i is selected. The GT policy selects the queue that maximizes the quantity $(\mu_i/(\lambda + \mu_i))^{1+n_i}$.

Previous work has suggested that the SED policy is an adequate policy for these multiqueue systems. Banawan and Zahorjan [2] compute the optimal policy for several models using numerical policy iteration and find that over 90 percent of the scheduling decisions are consistent with the SED policy.



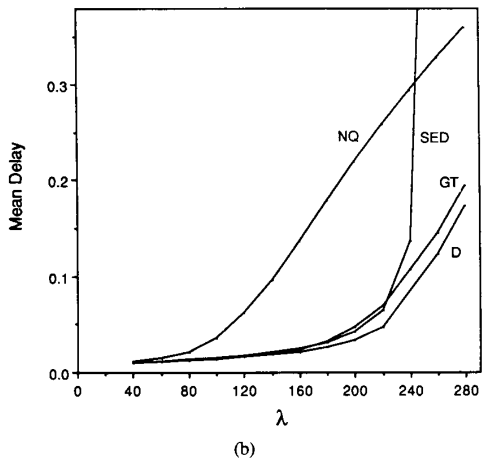


Fig. 5. The performance of the SED, NQ, GT, and D policies for the parallel queue model. The job arrival rate is λ. The curves are derived from a computer simulation of the system. (a) A system with three classes of servers: $\mu_{\text{fast}} = 16$ $m_{\text{fast}} = 5$, $\mu_{\text{med}} = 4$ $m_{\text{med}} = 20$, and $\mu_{\text{slow}} = 1$ $m_{\text{slow}} = \infty$. (b) A system with servers of speeds 128, 64, 32, 16, 8, 4, 2, and an infinite number of servers of speed 1.

Simulations by Rosberg and Kermani [22], [23] demonstrated that the SED policy was superior to the two other heuristic policies they considered. Furthermore, simulation studies by Houck [6] on a similar model (where each queue was served by a team of identical servers, but the queues could have different sized teams) indicated that the SED policy provided close-to-optimal delay. The results of our simulations (see Fig. 5) suggest a different conclusion: as in the single queue case, the SED policy is significantly suboptimal for some systems.

We have no simple solvable parallel queue model; the analogous two-speed model is intractable due to the huge state space (instead of a single queue length, there are queue lengths for each server). However, the infinite system limit is still tractable. The results here are essentially identical to the single queue case, with NQ and GT asymptotically optimal and SED giving significantly suboptimal delays. With no other analytical results for this model, we use our previous single queue results to motivate a new heuristic policy for the parallel queue model. The first step is to define a transcription between the parallel queue model and the single queue model.

Consider first a two-speed case, with an infinite number of slow servers and m fast servers (now, each with their own queue). Our central approximation here is to reduce this problem to a corresponding single queue problem. Whenever we have a nontrivial scheduling decision to make (that is when all of the fast servers are occupied), we use a modified threshold policy that treats the fast servers as if they had a single queue by adding up all of their queue lengths and then applying a threshold to this sum. A job is sent to the fast server with the shortest queue as long as the sum of the queue lengths of the fast servers is below some threshold; otherwise, the job is sent to an open slow server. Using the value for the threshold calculated from the single queue formula (3.3), we find that this policy performs within 2 percent of the optimal modified threshold policy over the entire range of ρ values. (The optimal modified threshold policy was determined through exhaustive simulation; see [28].) The delay values of the single queue and the parallel queue models are not the same; however, the preceding result suggests that the optimal thresholds of the two models are closely related.

Emboldened by the success of modeling the parallel system with the single queue system, we return to the case of general server speeds. Exploiting the similarity in thresholds between the single queue and parallel queue models, we apply the single queue/general speed deterministic D threshold to the parallel queue case. We treat all of the servers that are faster than the fastest open server as belonging to a single queue, and make the decision of whether or not to queue based on this threshold. A job to be queued is sent to the queue with the minimum $(n_i + 1)/\mu_i$.

B. Simulations

We consider the same two sets of server speeds as we did for the single queue model. Fig. 5(a) depicts the performance of the various policies for the first system configuration and Fig. 5(b) shows the same for the second system configuration. Both of these graphs are very similar to ones obtained for the corresponding single queue systems, providing further support for the approximation of modeling the parallel queues as a single queue. As in the single queue case, the naive SED policy performs poorly compared to the GT and D policies; the NQ policy is quite good for the first system, but is bad for the second system. Our D policy again outperforms all of the other policies over the entire range of arrival rates for both systems studied.

There are two other policies that we did not include in our graphs but that deserve mention. Chow and Kohler [5] devised a policy (which they conjectured is optimal) that is very similar to the GT policy, except that it maximizes the throughput *rate* before the next job arrival. Through simulations, we find that it performs slightly less well than GT. Another policy is that devised by Krishnan [11], based on a single application of policy iteration to a Bernoulli split random assignment algorithm. This policy does significantly better than SED, but performs worse than GT in our simulations. Furthermore, it displays suboptimal performance for our two-speed model in the large system limit [28].

VIII. Summary

We have studied the problem of controlling a queue served by a heterogeneous set of servers. While the general problem remains intractable, we identified a simplified version of the problem, the two-speed model, that is solvable. This two-speed model was used to analyze the heuristic policies of *shortest expected delay*, *never queue*, and *greedy throughput*. Despite its apparent naturalness, the SED policy performed significantly suboptimally in the limit of large systems. The other two policies performed suboptimally in small systems with extreme heterogeneity. Using the two-speed model as a basic source of insight, the D heuristic policy was introduced. It outperformed all of the other heuristics on the two system configurations we simulated, and was within 1/2 percent of the optimal threshold policy for the first system configuration. We then considered the related problem of parallel queues. Again, the analog of the single queue D policy outperformed all of the other policies.

Appendix
Calculation of Cost Functions

Recall that the cost function $C_\tau^{\text{fast}}(x)$ is defined by

$$C_\tau^{\text{fast}}(x) \equiv \lim_{t\to\infty} [T_\tau(x, t) - T_\tau(x-1, t)] \qquad (A.1)$$

where $T_\tau(x, t)$ is the total delay accumulated up to time t. When computing $C_\tau^{\text{fast}}(x)$ there are two cases to consider: $x \geq \tau$ and $x < \tau$. We will first consider $x \geq \tau$ and observe that in this case accepting a job merely induces a delay until the queue reverts back to its original length (the threshold policy sends all jobs to the slow servers until the queue length is below threshold). Let s be the time until one of the current jobs finishes; the probabilty distribution of the values of s is given by $f(s) = m\mu_{\text{fast}}e^{-m\mu_{\text{fast}}s}$. The total extra delay accumulated during this period is the delay in the queue, $s(x+m)$, plus the expected number of jobs sent to the slow servers their expected delay, $s\lambda/\mu_{\text{slow}}$. Thus, for $x \geq \tau$,

$$T_\tau(x, t) = \int_0^\infty dsf(s)[T_\tau(x-1, t-s) + s(x+m) + sR\lambda]. \qquad (A.2)$$

Substituting this expression into the definition (A.1), we find

$$C_\tau^{\text{fast}}(x) = \frac{x+m}{m\mu_{\text{fast}}} + \rho\left(\frac{1}{\mu_{\text{slow}}} - D_\tau\right). \qquad (A.3)$$

We now turn to the computation of $C_\tau^{\text{fast}}(x)$ when $x < \tau$, where the situation is a bit more complicated. Upon accepting a job, two events can happen; with probability $m\mu_{\text{fast}}/(\lambda + m\mu_{\text{fast}})$ the next event will be that one of the jobs currently being processed will finish, and with probability $\lambda/(\lambda + m\mu_{\text{fast}})$ the next event will be the arrival of a new job, which is then placed onto the queue. We can write the equation

$$\begin{aligned}T_\tau(x, t) = {} & (x+m)\int_0^\infty dsf(s)s \\ & + \frac{m\mu_{\text{fast}}}{\lambda + m\mu_{\text{fast}}}\int_0^\infty dsf(s)T_\tau(x-1, t-s) \\ & + \frac{\lambda}{\lambda + m\mu_{\text{fast}}}\int_0^\infty dsf(s)T_\tau(x+1, t-s).\end{aligned} \qquad (A.4)$$

Subtracting $T_\tau(x, t)$ from both sides, taking the limit $t \to \infty$, and rearranging terms, we find a recursion relation for $x < \tau$

$$C_\tau^{\text{fast}}(x) = \frac{x+m}{m\mu_{\text{fast}}} + \rho(C_\tau^{\text{fast}}(x+1) - D_\tau). \qquad (A.5)$$

References

[1] A Agrawala, E. Coffman, M. Garey, and S. Tripathi, "A stochastic optimization algorithm minimizing expected flow times on uniform processors," *IEEE Trans. Comput.*, vol. C-33, pp. 351-356, 1984.

[2] S. Banawan and J. Zahorjan, "Load sharing in heterogeneous queueing systems," in *Proc. IEEE INFOCOM '89*, 1988, pp. 731-739.

[3] C. Bell and S. Stidham, "Individual versus social optimization in the allocation of customers to alternative servers," *Management Sci.*, vol. 29, pp. 831-839, 1983.

[4] F. Bonomi and A. Kumar, "Adaptive optimal load balancing in a heterogeneous multiserver system with a central job scheduler," in *Proc. IEEE 8th Int. Conf. Distrib. Comput. Syst.*, 1988, pp. 500-508.

[5] Y. Chow and W. Kohler, "Models for dynamic load balancing in a heterogeneous multiple processor system," *IEEE Trans. Comput.*, vol. C-28, pp. 354-361, 1979.

[6] D. Houck, "Comparison of policies for routing customers to parallel queueing systems," *Oper. Res.*, vol. 35, no. 2, pp. 306-310, 1987.

[7] R. Howard, *Dynamic Programming and Markov Processes*. Cambridge, MA: MIT Press, 1960.

[8] D. L. Jagerman, "Some properties of the Erlang loss function," *Bell Syst. Tech. J.*, vol. 53, pp. 525-551, 1974.

[9] F. P. Kelly, "Routing in circuit switched networks: Optimization, shadow prices and decentralization," *Advances Appl. Probability*, vol. 20, pp. 112-144, 1988.

[10] L. Kleinrock, *Communication Nets*. New York: McGraw-Hill, 1964, pp. 125-126.

[11] K. R. Krishnan, "Joining the right queue and routing in data networks," Bellcore Tech. Memo.; also see "Joining the right queue: A Markov decision rule," in *Proc. IEEE Conf. Decision Contr.*, 1987, pp. 1863-1868.

[12] —, "Adaptive routing for telephone traffic," Bellcore Tech. Memo., 1987.

[13] P. Kumar and J. Walrand, "Individually optimal routing in parallel systems," *J. Appl. Probability*, vol. 22, pp. 989-995, 1985.

[14] R. Larsen and A. Agrawala, "Control of a heterogeneous two-server exponential queueing system," *IEEE Trans. Software Eng.*, vol. SE-9, pp. 522-526, 1983.

[15] W. Lin and P. Kumar, "Optimal control of a queueing system with two heterogeneous servers," *IEEE Trans. Automat. Contr.*, vol. AC-29, pp. 696-703, 1984.

[16] S. Lippman and S. Stidham, "Individual versus social optimization in exponential congestion systems," *Oper. Res.*, vol. 25, pp. 233-247, 1977.

[17] P. Naor, "On the regulation of queue size by levying tolls," *Econometrica*, vol. 37, pp. 15-24, 1969.

[18] R. Nelson and D. Towsley, "On maximizing the number of departures before a deadline on multiple processors," IBM Rep. RC 11255, 1985.

[19] ——, "Comparison of threshold scheduling policies for multiple server systems," IBM Rep. RC 11256, 1985.
[20] G. F. Newell, *Approximate Stochastic Behavior of n-Server Service Systems with Large n.* Berlin, Germany: Springer-Verlag, 1973.
[21] R. Righter, "The stochastic sequential assignment problem with random deadlines," *Probability Eng. Inform. Sci.*, vol. 1, pp. 189-202, 1987.
[22] Z. Rosberg and P. Kermani, "Customer routing to different servers with complete information," IBM Rep. RC 13765, 1988.
[23] ——, "Customer routing to different servers with complete information," in *Proc. Twenty-Sixth Annu. Allerton Conf. Commun., Contr., Comput.*, 1988, pp. 566-568.
[24] S. Shenker and A. Weinrib, "Asymptotic analysis of large heterogeneous queueing systems," in *Proc. Sigmetrics 88*, 1988, pp. 56-62.
[25] S. Stidham, "Optimal control of admission to a queueing system," *IEEE Trans. Automat. Contr.*, vol. AC-30, pp. 705-713, 1985.
[26] ——, "Scheduling, routing, and flow control in stochastic networks," Univ. of North Carolina Tech. Rep. UNC/ORSA/TR-86/22, 1986.
[27] J. Walrand, "A note on 'Optimal control of a queueing system with heterogeneous servers,'" *Syst. Contr. Lett.*, vol. 4, pp. 131-134, 1984.
[28] A. Weinrib and S. Shenker, "Greed is not enough: Adaptive load sharing in large heterogeneous systems," in *Proc. IEEE INFOCOM '88*, 1988, pp. 986-994.
[29] T. Yum and M. Schwartz, "The join-biased-queue rule and its application to routing in computer communication networks," *IEEE Trans. Commun.*, vol. COM-29, pp. 505-511, 1981.

Scott Shenker received the Sc.B. degree in physics from Brown University, Providence, RI, in 1978 and the Ph.D. degree in physics from the University of Chicago, Chicago, IL, in 1983. His graduate studies focused on the scaling properties of chaotic dynamical systems.

After a year of postdoctoral work at Cornell University, Ithaca, NY, he joined the Xerox Palo Alto Research Center where he is now a member of the Computer Science Laboratory. His current research interests include computer systems modeling, game-theoretic approaches to resource allocation, and the dynamics of computer networks.

Abel Weinrib received the S.B. degree from the Massachusetts Institute of Technology, Cambridge, in 1979 and the Ph.D. degree from Harvard University, Cambridge, MA, in 1983, both in physics.

He is currently a member of the Technical Staff in the Applied Research Area at Bellcore, where he has been since 1985. His current research interests include control issues in circuit and packet networks and algorithms for distributed systems.

Dr. Weinrib is a member of the Association for Computing Machinery, the IEEE Computer Society, Phi Beta Kappa, and Sigma Xi.