Analyzing the Effect of Prioritized Background Tasks in Multiserver Systems

Adam Wierman[1] Takayuki Osogami[2] Mor Harchol-Balter[3]
Alan Scheller-Wolf[4]

December 2003

CMU-CS-03-213 3

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[1]Carnegie Mellon University, Computer Science Department. Email: acw@cs.cmu.edu.
[2]Carnegie Mellon University, Computer Science Department. Email: osogami@cs.cmu.edu.
[3]Carnegie Mellon University, Computer Science Department. Email: harchol@cs.cmu.edu.
[4]Carnegie Mellon University, Computer Science Department. Email: awolf@andrew.cmu.edu.

Keywords: reliability, M/GI/k, multiserver queue, priority queue, server farm, matrix analytic methods, busy periods, multi-class queue, preemptive priority.

Abstract

Computer systems depend on high priority background processes to provide both reliability and security. This is especially true in multiserver systems where many such background processes are required for data coherence, fault detection, intrusion detection, etc. From a user's perspective, it is important to understand the effect that these many classes of high priority, background tasks have on the performance of lower priority user-level tasks.

We model this situation as an M/GI/k queue with m preemptive-resume priority classes, presenting the first analysis of this system with more than two priority classes under a general phase-type service distribution. (Prior analyses of the M/GI/k with more than two priority classes are approximations that, we show, can be highly inaccurate.) Our analytical method is very different from the prior literature: it combines the technique of dimensionality reduction [10] with Neuts' technique for determining busy periods in multiserver systems [21], and then uses a novel recursive iteration technique. Our analysis is approximate, but, unlike prior techniques, can be made as accurate as desired, and is verified via simulation.

1 Introduction

Machines on the Internet are increasingly prone to malicious attacks from viruses, identity thieves, and other dangers. To protect users from malicious attacks as well as system faults, errors, and failures resulting from system complexity, computers today typically run an array of background processes for the purposes of fault recovery, fault isolation, fault masking, intrusion detection, virus checking, etc. (See [1] for a survey of such tasks.) Many of these processes rely on preemptive error detection, where service delivery is suspended while checks for latent errors and dormant faults are performed. Thus, in modern systems user-level tasks may be preempted by a wide range of higher priority, dependability tasks.

Examples of high priority dependability tasks are abundant. In particular, significant research has been performed for the purposes of system rollback [5, 16], where checkpointing is performed so that at the detection of a fault or error the system can be rolled back to a stable state. Further, many coherence processes in distributed systems depend on background tasks for purposes such as synchronizing clocks and ordering communications [15, 17]. A third example of background processes needed for dependability are exception handlers, which become important in distributed systems [27]. Finally, a fundamental problem in distributed dependable systems is that of reaching agreement on the identity of correctly functioning processors. Thus, background processes must be designed to reach agreement in the presence of faults [2, 26].

Given the necessity of such dependability processes, it is important to understand the impact of these high priority background tasks on the performance of lower priority user-level tasks, and also how this impact can be reduced. Since the 1950's queueing theorists have studied the effect of high priority tasks on lower priority tasks in the context of an M/GI/1 queue. Over the past couple decades, multiserver systems (server farms) have increased in popularity due to their low cost, versatility, and reliability benefits. As server farms have become more ubiquitous, queueing theory has shifted to studying the effect of priorities in multiserver systems with the aim of improving the design of these systems.

Unfortunately, for multiserver systems, performance results under prioritization have been quite limited. Only the case of two priority classes with exponential service times has been analyzed. This is insufficient, as in practice there are many different levels of high priority dependability tasks, requiring more than two priority classes, and job size distributions are often not exponential. Yet, for this case of more than 2 priority classes, or for general service times, only ad-hoc approximations exist. Worse still, the accuracy of these existing approximations has never been established.

This paper provides the first analytical method to evaluate mean class delay[1] in an M/GI/k system (i) with an arbitrary number of preemptive priority classes and (ii) where the job size distribution, G, is a phase-type (PH) distribution[2]. Neither (i) nor (ii) above have ever been addressed in the prior literature, aside from

[1]The delay of a job is the time from when a job arrives until it leaves, minus the time the job spends in service.

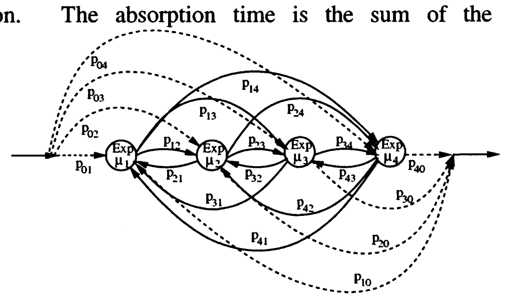
[2]The set of PH distributions is dense in the set of all non-negative general distributions. A PH distribution is the distribution of the absorption time in a finite state continuous time Markov chain. The figure shows a 4-phase PH distribution, with $n = 4$ states, where the ith state has exponentially-distributed sojourn time with rate μ_i. With

the ad-hoc approximations mentioned above. What makes analysis of this problem difficult for an arbitrary number of priority classes, m, is that the state space of the system grows infinitely in m dimensions, as one needs to track the number of jobs in each priority class. To overcome this problem, we introduce a new analytical approach, called Recursive Dimensionality Reduction (RDR), that allows us to recursively reduce this m-dimensionally infinite state space to a 1-dimensionally infinite state space. An important element of RDR involves approximating a sequence of complex busy periods; RDR is approximate only in that we compute a finite number of moments (three) of these busy period durations. As we will show, the RDR method is quite accurate when using three moments, and its accuracy can be increased arbitrarily by computing more moments.

In theory RDR can handle systems with any number of servers, any number of priority classes, and PH service times. Practically however, the RDR method increases in complexity with both the number of servers k and the number of classes m. Nevertheless, for all the scenarios explored in this paper, the computation time under RDR is less than 0.1 sec. Because RDR becomes less practical under high m and k, we develop a much simpler, but still accurate, approximation RDR-A. RDR-A simplifies calculations by approximating an m priority system with a two priority system, which is then solved using RDR — RDR is much easier to implement in this special case.

The contributions of this paper are four-fold. First, we introduce the RDR technique, providing the first analysis of an M/GI/k system with arbitrary numbers of priorities. Second, we provide a thorough evaluation of existing m class M/GI/k approximation formulas by Buzen & Bondi [3, 4] and by Mitrani & King [20] for the mean delay. Buzen & Bondi assume that the effect of priorities in multiserver systems is well-approximated by the effect of priorities in a single-server system, while Mitrani & King aggregate all the higher priority classes into a single exponential class. Both these approximations are coarse; we will show exactly when they can lead to significant error. The third contribution of the paper is our new approximation algorithm RDR-A, which is both computationally efficient and greatly improves upon the accuracy of prior approximations. The fourth contribution of this paper is a characterization of the behavior of multiserver systems with priorities. We find this to be very different from a single-server system, and we contribute a set of design guidelines for setting up (high priority) background processes in a way that minimizes the unwanted delay on the (lower priority) user tasks in multiserver systems. For example, we

probability p_{0i} we start in the ith state, and the next state is state j with probability p_{ij}. Each state has some probability of leading to absorption. The absorption time is the sum of the times spent in each of the states.



find that tasks that require more deterministic computation times are much less painful to lower priority tasks than tasks that are highly variable.

2 Prior work

A large number of papers have been written on the analysis of mean delay in an M/GI/k queue with priority classes. However, most of these papers are restricted to only two priority classes under the additional constraint that the service distributions are exponential. The few papers that deal with $m > 2$ priority classes, are coarse approximations based on assuming that the multiserver behaves like a single server system or assuming identical job size distributions for all classes.

2.1 Approximations for an M/GI/k queue with m $>$ 2 classes

Two types of approximations have been proposed in literature. Buzen and Bondi's approximation (which we denote by BB) was originally derived for exponential job size distributions [4] and was later applied to general job size distributions [3]. The other approximation (which we denote by MK-N) is due to Mitrani and King [20], and also used by Nishida [23] to extend the latter author's analysis of two priority classes to $m > 2$ priority classes. MK-N is restricted to exponential job size distributions.

The Buzen-Bondi (BB) approximation

The BB approximation is based on an intuitive observation that the "improvement" of priority scheduling over FCFS scheduling under k servers is similar to that for the case of 1 server:

$$\frac{E[D^{\text{M/GI/}k\text{/prio}}]}{E[D^{\text{M/GI/}k\text{/FCFS}}]} \approx \frac{E[D^{\text{M/GI/1/prio}}]}{E[D^{\text{M/GI/1/FCFS}}]} = \text{scaling factor.} \tag{1}$$

Here $E[D^{\text{M/GI/}k\text{/prio}}]$ is the overall mean delay under priority scheduling with k servers of speed $1/k$, and $E[D^{\text{M/GI/}k\text{/FCFS}}]$ is defined similarly for FCFS. This relation is exact when job sizes are exponential with the same rate for all classes, however it is unclear what happens when this is not the case.

BB analyzes the mean delay of the lowest priority class ($E[D_L]$) by analyzing both the mean delay over all classes ($E[D]$) and the mean delay of the higher priority classes ($E[D_H]$) and then using the following relation: $E[D] = \frac{\lambda_H}{\lambda_H+\lambda_L}E[D_H] + \frac{\lambda_L}{\lambda_H+\lambda_L}E[D_L]$, where λ_H and λ_L are the arrival rates of the higher and lowest priority jobs respectively. In analyzing the mean delay of multiple classes, BB aggregates all the relevant classes into one class and analyzes the corresponding M/GI/k/FCFS queue. It then calibrates this result using the scaling factor in (1) in order to capture the effect of prioritization. Namely, $E[D^{\text{M/GI/}k\text{/prio}}] \approx E[D^{\text{M/GI/}k\text{/FCFS}}]\frac{E[D^{\text{M/GI/1/prio}}]}{E[D^{\text{M/GI/1/FCFS}}]}$.

We now provide intuition as to why BB might lead to error in predicting the mean delay. Consider two classes, high priority (H) and low priority (L), where class H jobs have smaller mean size. Priority scheduling allows small jobs to avoid waiting behind the long jobs, leading to significant decrease in mean delay as compared to FCFS. However, having multiple servers also allows small jobs to avoid queueing behind the long jobs (especially if these long jobs are sparse), and thus the decrease in mean delay due to priority scheduling for the multiserver case may not be as significant as in the single server case.

The Mitrani-King-Nishida (MK-N) approximation

The MK-N approximation analyzes the mean delay of the lowest priority class in an M/M/k queue with $m \geq 2$ priority classes by aggregating all the higher priority classes. Thus, instead of aggregating all jobs into one class, as BB does, MK-N aggregates into two classes. The job size distribution of the aggregated class is then approximated with an exponential distribution by matching the first moment of the distribution. There are two sources of errors in MK-N. The first source of error, which we call *priority error*, results from ignoring the priority scheduling among the higher priority classes. The second source, which we call *distribution error*, results from the fact that the aggregated job size distribution matches only one moment.

Consider first the priority error. In the case of a single server, there is no priority error. This is because low priority jobs are excluded from service during busy periods of high priority classes, and the busy period duration of the higher priority classes is the same regardless of whether or not the higher priority classes are prioritized. In the case of multiple servers, however, priority error exists: low priority jobs are more likely to be present at the end of a busy period.

Next consider the distribution error. The distribution error exists even in the case of a single server unless the first two moments of the aggregated job size distribution are matched, as the mean delay of the lowest priority jobs, $E[D_L]$, depends on the first two moments of the job size distributions:

$$E[D_L] = \frac{\rho_H}{1-\rho_H} E[X_L] + \frac{\lambda E[X^2]}{2(1-\rho_H)(1-\rho)}, \tag{2}$$

where ρ_H is the load of high priority jobs, ρ is the overall load, $E[X_L]$ is the mean size of the lowest priority jobs, and $E[X^2]$ is the second moment of the overall job size distribution. In the case of multiple servers, matching even the first two moments is not sufficient in general. This can be gleaned from recent results illustrating the sensitivity of the stability conditions for the M/GI/k systems to the exact tail behavior of the service distribution [28, 31].

2.2 Analysis of an M/GI/k queue with m = 2 priority classes

In the case of two priority classes, exact numerical and algorithmic approaches have been proposed. Most approaches are restricted to exponential job size distributions [6, 7, 9, 19, 20], and the only work dealing with non-exponential service times (limited to hyperexponential distributions) is a paper, not yet published, by Sleptchenko et. al. [30]. There are a number of approximations based on aggregation or truncation of the state space [11, 12, 23, 14], sometimes in combination with the matrix analytic method and state space partitioning [22]. All the above papers assume preemptive-resume priorities (as in this paper), but there also are a few papers on non-preemptive priorities [8, 13, 29].

Exact analyses for two priority classes with exponential job size distributions either use (i) matrix analytic methods or (ii) generating function methods. Matrix analytic methods are unable to directly handle a two-dimensionally infinite state space; so in order to overcome this, Miller [19] partitions the state space into blocks and then "super-blocks," according to the number of high priority jobs in queue. This partitioning is quite complex and is unlikely to be generalizable to non-exponential job sizes. In addition, Miller experiences numerical instability issues when $\rho > 0.8$. All the other exact analyses of two priority classes under exponential job size distributions capitalize on the exponential job sizes directly by explicitly writing out the system of balance equations and then finding roots via generating functions. These techniques in general yield complicated mathematical expressions susceptible to numerical instabilities at higher loads. For example, Mitrani and King [20] report that for larger systems (eight servers) they begin to experience numerical stability problems – their solution yields some negative probabilities. Gail, Hantler, and Taylor [9] follow a similar approach and also report stability problems.

Finally, Sleptchenko et. al. [30] consider a two-priority, multiserver system where, within each priority class, there may be a number of subclasses, each with its own different exponential job size distribution. This is equivalent to assuming a hyperexponential job size distribution for each of the two priority classes. The problem is solved by writing the system balance equations and solving them iteratively. Unfortunately, their technique does not appear to generalize to distributions other than the hyperexponential distribution.

3 RDR analysis of M/GI/k with m priority classes

In this section, we describe our new analytic method: Recursive Dimensionality Reduction (RDR), which allows us to analyze the mean delay in an M/GI/*k*/FCFS queue having m preemptive-resume priority classes. In the case where there are only two priority classes and both classes have exponentially distributed service times, the RDR technique reduces to an already known technique, introduced in [25], simply called *dimensionality reduction,* which reduces a two dimensionally infinity state space to a one dimensionally infinite state space. However, when the number of priority classes is $m > 2$ or job sizes are PH distributed, simple dimensionality reduction does not apply.

In handling the $m > 2$ priority classes, the RDR method involves a recursive algorithm that uses the analysis of the $m - 1$ priority class case to analyze the m-th priority class. The recursive algorithm also requires utilizing Neuts' algorithm [21] for analyzing busy periods in multiserver systems. Our RDR method generalizes to PH job size distributions — this discussion is deferred to Appendix A.

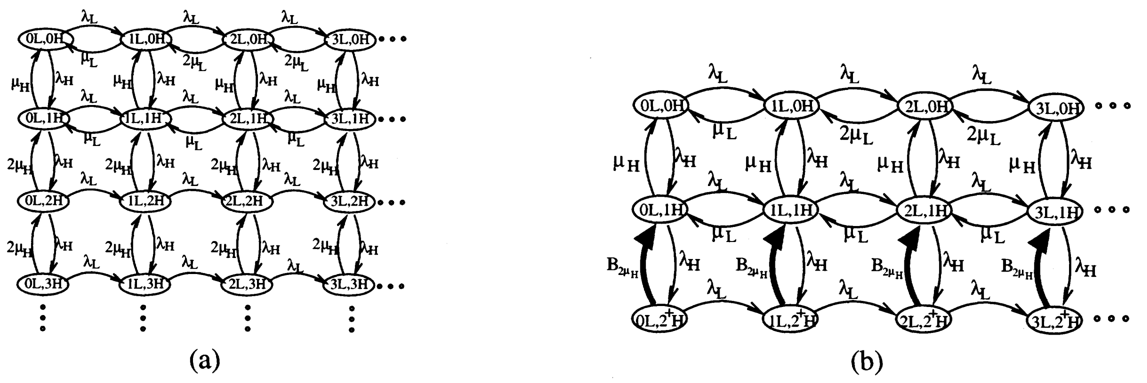


Figure 1: *Markov chain for a system with 2 servers and 2 priority classes where all jobs have exponential sizes. The chain in (a) is infinite in two dimensions. Via the Dimensionality Reduction technique, we arrive at the chain in (b), which uses busy period transitions, and is only infinite in one dimension.*

3.1 Two priority classes, exponential job sizes

We first consider the simple case of two servers and two priority classes, where both high and low priority jobs have exponentially distributed sizes with rates μ_H and μ_L respectively. For this case, the RDR method reduces to the simpler dimensionality reduction method of [25]. Figure 1 (a) illustrates a Markov chain of this system, whose states track the number of high priority and low priority jobs; hence this chain grows infinitely in two dimensions. Observe that high priority jobs simply see an M/M/2 queue, and thus their mean delay is well-known. Low priority jobs, however, have access to either an M/M/2, M/M/1, or no server at all, depending on the number of high priority jobs. Thus their mean delay is more complicated.

Figure 1 (b) illustrates the reduction of the 2D-infinite Markov chain to a 1D-infinite Markov chain via RDR. The 1D-infinite chain tracks the number of low priority jobs exactly. For the high priority jobs, the 1D-infinite chain only differentiates between zero, one, and two-or-more high priority jobs. As soon as there are two-or-more high priority jobs, a *high priority busy period* is started. During the high priority busy period, the system only services high priority jobs, until the number of high priority jobs drops to one.[3] The length of time spent in this high priority busy period is exactly an M/M/1 busy period where the service rate is $2\mu_H$. We denote the length of this busy period by the transition labeled $B_{2\mu_H}$. We use a PH distribution to match the first 3 moments of the distribution of $B_{2\mu_H}$.[4] The limiting probabilities in this 1D-infinite chain can be analyzed using the matrix analytic method, which in turn gives the mean delay via Little's law.

Figure 2 shows the generalization to a 3-server system. We simply add one row to the Markov chain

[3]Throughout the paper a "higher priority busy period" is defined as the time from when the system has k higher priority jobs until there are only $k-1$ higher priority jobs.

[4]Matching three moments of busy period distributions is often sufficient to guarantee accurate modeling of many queueing systems with respect to mean performance [24].

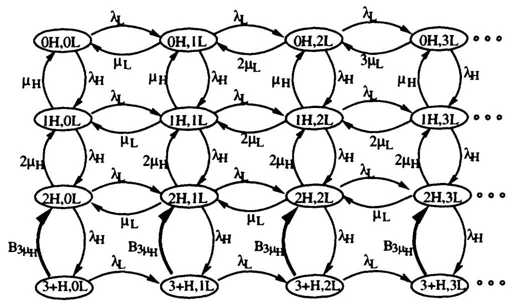


Figure 2: *This Markov chain illustrates the case of 2 priority classes and 3 servers.*

shown in Figure 1, and now differentiate between 0, 1, 2, or 3-or-more high priority jobs. This can be easily extended to the case of $k > 3$ servers.

RDR is far simpler and more computationally efficient than methods shown in the prior published literature for analyzing the mean delay in an M/M/k queue with dual priorities exponential job sizes. Furthermore, RDR allows more classes, allows general PH job sizes, and is highly accurate. (See Section 4 for validation.) By contrast, the prior literature on the M/M/k priority queue has sometimes resulted in numerically unstable computations [8, 9] or complicated partitions of the state space [19, 22].

3.2 m priority classes, exponential job sizes

Below we describe the RDR solution for systems with multiple servers and priority classes, beginning with the simpler case of two servers and three priority classes. We denote the 3 priority classes by high-priority (H), medium-priority (M), and low-priority (L). The mean delay for class H jobs and that for class M jobs are easy to compute. Class H jobs simply see an M/M/2 queue. Class M jobs see the same system that the low-priority jobs see in an M/M/2 queue having 2 priority classes. Replacing the L's by M's in the chain in Figure 1 yields the mean delay for the M class jobs.

The analysis of the class L jobs is more difficult. The obvious approach would be to aggregate the H and M jobs into a single class, so that we we have a 2-class system (H-M and L jobs). Then we could apply the RDR technique of the previous section, tracking exactly the number of low-priority jobs and maintaining limited state on the H-M class. This is the approach that we follow in Section 5 in deriving our RDR-A approximation. However, the above approach is imprecise because the duration of the busy periods in the H-M class depend on whether the busy period was started by 2H jobs, 1H and 1M job, or 2M jobs in service. By ignoring the priorities among H's and M's, we are ignoring the fact that some types of busy periods are more likely. Even given the information on who starts the busy period, that still does not suffice to determine its duration, because the duration is also affected by whether the H-M class has internal prioritization.

Thus a precise delay analysis of class L requires tracking more information. We need to track whether there are zero, one, or at least two H-M jobs. Given that there is only one H-M job, we need to know whether it is of type H or type M. Given that there are at least two H-M jobs, we are in an H-M busy period. Thus, we

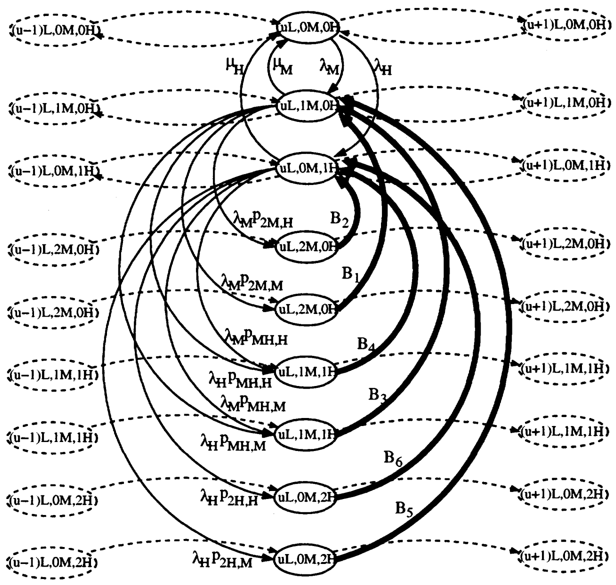


Figure 3: *1D-infinite chain used to compute mean delay for low-priority jobs in the case of 3 priority classes and 2 servers, and all exponential service times. The six busy period transitions for this chain will be computed from the chain in Figure 1 (b).*

need to know whether the busy period was started by 2 H jobs, 2 M jobs, or 1H and 1M job. We also need to know whether the busy period is ended by an H job or an M job. For an M/M/2 with 3 priority classes, six types of busy periods are possible, depending on the classes of jobs that start the busy period and the class of the job that is left at the end of the busy period. We derive the busy period duration by first conditioning on who ends the busy period, which is stochastically equivalent to the unconditioned approach.

Figure 3 shows the level of the 1D infinite Markov chain, where the number of class L jobs is u. In state (uL,vM,wH), v class M jobs and w H class jobs are in system if $v+w < 2$; otherwise, the state (uL,vM,wH) denotes that we are in a H-M busy period that was started by v class M jobs and w class H jobs. When the Markov chain is in state (uL,0M,0H), an arrival of an M class job triggers a transition to state (uL,1M,0H). In state (uL,1M,0H), an arrival of an M class job triggers a transition to state (uL,2M,0H). Observe that both states in the fourth and fifth row are labeled (uL,2M,0H). In both states, two servers are working on jobs of class M or H, and this busy period is started by two M class jobs. The two states labelled (uL,2M,0H) differ in the class of the job that is left at the end of the H-M busy period. In state (uL,2M,0H) of the fourth row, the busy period ends leaving a class H job, and in state of the fifth row, the busy period ends leaving a class M job. (In our Markov chain, the class of job left at the end of a busy period is probabilistically determined at the beginning of the busy period and the duration of the busy period is conditioned on the class of the job left at the end.) Other transitions are defined analogously. The six types of busy periods are labeled B_1, B_2, ..., B_6; the durations of these busy periods are approximated by PH distributions that match the first three moments. Finally, $p_{2M,H}$ ($p_{MH,H}$ and $p_{2H,H}$, respectively) denotes the probability that the busy period started by two M class jobs (one M class job and one H class job, and two H class jobs, respectively) ends leaving one H class job; $p_{2M,M}$, $p_{MH,M}$, and $p_{2H,M}$ are defined analogously.

It remains to determine how to analyze the moments of the duration of busy periods, $B_1, B_2, ..., B_6$, and probabilities $p_{2M,M}, p_{2M,H}, p_{MH,M}, p_{MH,H}, p_{2H,M}$, and $p_{2H,H}$ in Figure 3. Observe that the Markov chain for class H and M jobs (Figure 1 (b)) is a QBD process, and the duration of each busy period in Figure 3 corresponds to the first passage time from a state in level 1 to a state in level 0 in Figure 1 (b). Likewise, each probability in Figure 3 corresponds to the probability that a certain state is the first state in level 0 given that we start at a certain state in level 1 of Figure 1 (b). All of these quantities can be calculated using Neuts' algorithm [21]. We provide a the precise description of how Neuts' algorithm is applied in Appendix B.

The extension of RDR to $m > 3$ classes is trivial. For example, for the case of $m = 4$ classes, we create a version of the Markov chain in Figure 3 done for 4 classes. This chain tracks exactly the number of jobs in class 4, and creates busy periods for the three higher priority classes. To derive the busy periods for the three higher priority classes, we make use of the Markov chain in Figure 3 and compute the busy periods in that chain using Neuts' algorithm. Note that for an m-class system with k servers, there are $\binom{m+k-2}{k}\binom{m+k-3}{k-1}$ possible busy periods. That is, the number of different types of busy periods is polynomial in k if m is constant (k^{2m-4}), and it is polynomial in m if k is constant (m^{2k-1}); however, it is exponential in k and m if neither k nor m is constant.

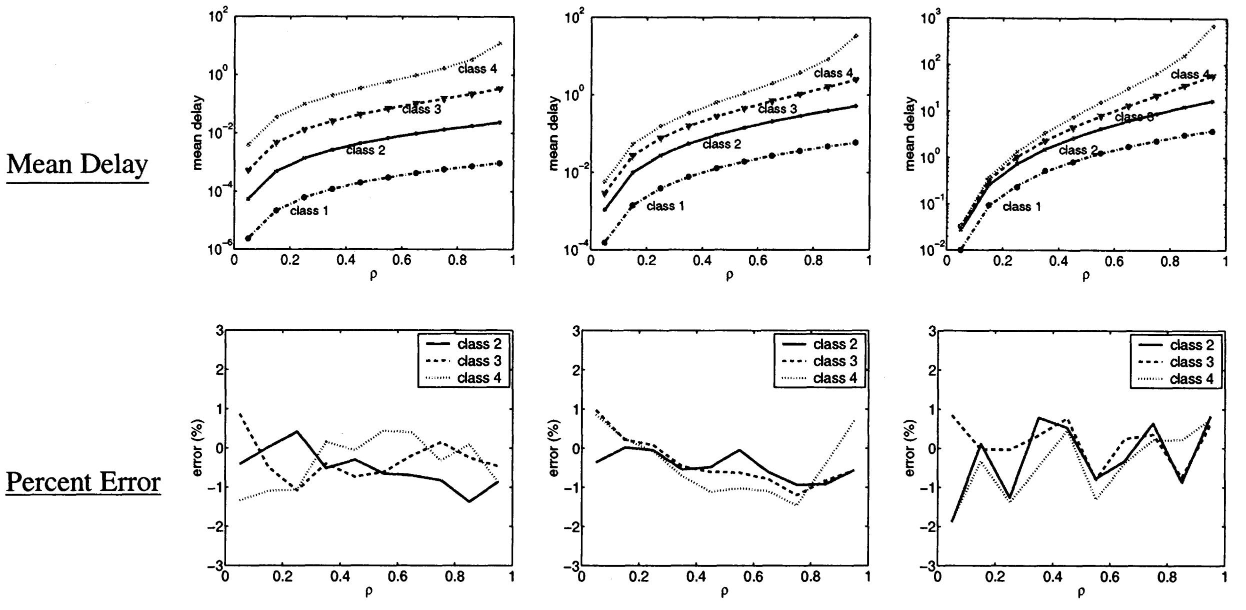
RDR can also handle PH job size distributions. In this case class H is easily modeled as a QBD process. To analyze the mean delay of class M jobs, we need to analyze the busy period of class H jobs, which is is no longer a busy period in the corresponding M/M/k queue. We thus must use Neuts' algorithm to analyze this busy period. Lower priority classes are analyzed recursively as described above. Details of the analysis of case of PH job size distributions are provided in Appendix A.

4 Validation of RDR

We now validate RDR against simulation. We show only a subset of the wide range of validations we performed. In all cases, the mean delay predicted by RDR is within 2% of the simulated value.

Figure 4 shows a validation of RDR in the case of an M/M/2 queue with four priority classes, where class 1 has the highest priority and class 4 has the lowest priority. The load of each class is one-quarter of the total load, i.e. each class has the same load. The mean job size of class i is set $E[X_i] = \gamma^{4-i}$. Namely, $\gamma < 1$ implies small high priority jobs, and $\gamma > 1$ implies large high priority jobs. The three columns in Figure 4 correspond to different values of γ: 1/4, 1, and 4. Figure 5 shows the validation of RDR for an M/GI/2 queue with two priority classes. Here we assess the effect of C^2, the squared coefficient of variability of the job size distribution, defined as the variance divided by the square of the mean.

The plots in the top row of Figure 4 show the mean delay under both RDR and simulation for each class as a function of total load, ρ, in log scale. The plots in the top row of Figure 5 also show the mean delay,



(a) High prio.: small ($\gamma = 1/4$) (b) All same mean ($\gamma = 1$) (c) High prio.: large ($\gamma = 4$)

Figure 4: *Validation of RDR against simulation for the case of M/M/2 with 4 priority classes, where the mean job sizes are $E[X_4] = 1$, $E[X_3] = \gamma$, $E[X_2] = \gamma^2$, $E[X_1] = \gamma^3$, and the load of class i is $\rho_i = \rho/4$ for each i. The three columns show three cases of γ's: 1/4, 1, and 4. The top row shows the mean delay given by RDR (shown in lines) and the mean delay in simulation (shown in dots) in log scale as a function of total load, ρ. The bottom row shows the relative difference between RDR and simulation. Class 1 is not shown because its analysis is by definition exact.*

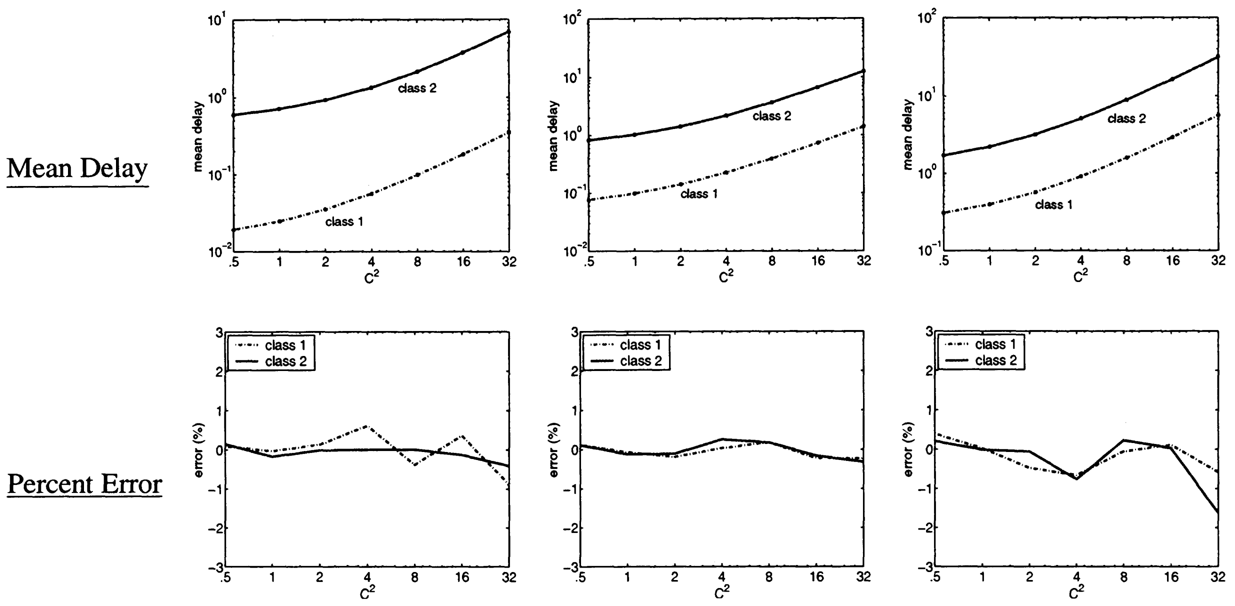
however as a function of C^2. From the top row of Figure 4 and 5, it appears that analysis matches simulation perfectly across all ρ's, and for all γ's under both exponential and general service times.

Taking a closer look, the plots in the bottom row of Figures 4 and 5 show the relative error in the mean delay, where relative error is defined as

$$\text{error} = 100 \times \frac{(\text{mean delay by RDR}) - (\text{mean delay by simulation})}{(\text{mean delay by simulation})} \quad (\%).$$

We see that the relative error in the mean delay of RDR compared to simulation is within 2% for all classes, for all ρ's, and for all γ's, and within 1% for most cases. On average, RDR tends to underestimate the mean delay by 0.2% of the simulation.

Finally, we note that RDR is much more computationally efficient than simulation. For each evaluation, a simulation is run 30 times, and in each run 1,000,000 events are generated. In this setting, simulation takes a day to generate each figure, while our analysis takes only a few minutes. Further, we need to generate 1,000,000 events to obtain the accuracy shown. For the case of 100,000 events, we see five times as much inaccuracy. Thus, we conjecture that as we increase the number of events in simulation, the difference in our analysis and the simulation will decrease even further.



(a) High prio.: small ($\gamma = 1/4$) (b) All same mean ($\gamma = 1$) (c) High prio.: large ($\gamma = 4$)

Figure 5: *Validation of RDR against simulation for the case of M/GI/2 with 2 priority classes, where the mean job sizes are* $E[X_2] = 1$, $E[X_1] = \gamma$, *and the total load is* $\rho = 0.6$ *and load is balanced between the classes. The three columns show three cases of* γ*'s: 1/4, 1, and 4. The top row shows the mean delay given by RDR (shown in lines) and the mean delay in simulation (shown in dots) in log scale as a function of total load,* ρ. *The bottom row shows the relative difference between RDR and simulation.*

5 RDR-A approximation for M/GI/k with m priority classes

We have seen that the RDR analysis method is highly accurate. However this method can be computationally intensive. This motivates us to introduce an approximation based on RDR called RDR-A. RDR-A applies to $m \geq 2$ priority classes and PH job size distributions. In Section 5.1 we define RDR-A, and in Section 5.2 we compare its accuracy against the existing approximations in the literature. We find that throughout RDR-A predicts results within 5% accuracy, while BB and MK-N can have errors of 50% or more.

5.1 Introducing RDR-A

The key idea behind RDR-A is that the RDR computation is far simpler when there are only two priority classes: H and L. In RDR-A, under m priority classes, we simply aggregate these classes into two priority classes, where the $m-1$ higher priority classes become the new aggregate H class and the mth priority class becomes the L class. We define the H class to have a PH job size distribution that matches the first three moments of the aggregation of the $m-1$ higher priority classes.

Observe that the RDR-A method is very similar to the MK-N approximation. The only difference is that in MK-N, both the H and L classes are exponentially-distributed. Thus under MK-N, the H class only matches the *first* moment of the aggregate $m-1$ classes, whereas under RDR-A *three* moments are matched. The reason that we are able to match the first three moments, rather than just the first moment is that we have the RDR technique, which allows the analysis multiserver priority queues with PH job size distributions.

5.2 Comparing RDR-A with BB and MK-N

We now compare the accuracy of RDR-A with the two existing approximations in the literature: BB and MK-N. In our comparison we assume $k = 2$ servers and $m = 4$ priority classes. We consider both the case where each priority class has an exponential job size distribution $C^2 = 1$ (top half of Figure 6) and the case of PH job size distributions where $C^2 = 8$ (bottom half of Figure 6). Each class may have a different mean job size, and these are chosen to vary over a large range, determined by parameter γ. The mean job size of class i is set $E[X_i] = \gamma^{4-i}$. Namely, $\gamma < 1$ implies small high priority jobs. We equalize the load between the classes, i.e., $\rho_i = \rho/4$, where ρ_i is the load of class i. Mean delay is evaluated for each class of jobs.

The error of an approximation is defined as follows:

$$\text{error} = 100 \times \frac{\text{(mean delay by approximation)} - \text{(exact mean delay)}}{\text{(exact mean delay)}} \quad (\%).$$

Thus, positive error means overestimation and negative error means underestimation of the approximation. In the case where $C^2 = 1$ for each class, we compare the results of the approximations against the results of RDR. This is possible because RDR is highly accurate for this case (within 1-2%). In the case where $C^2 = 8$ for each class, we compare the results of the approximations against simulation, because RDR is less accurate in this case (see Section 4).

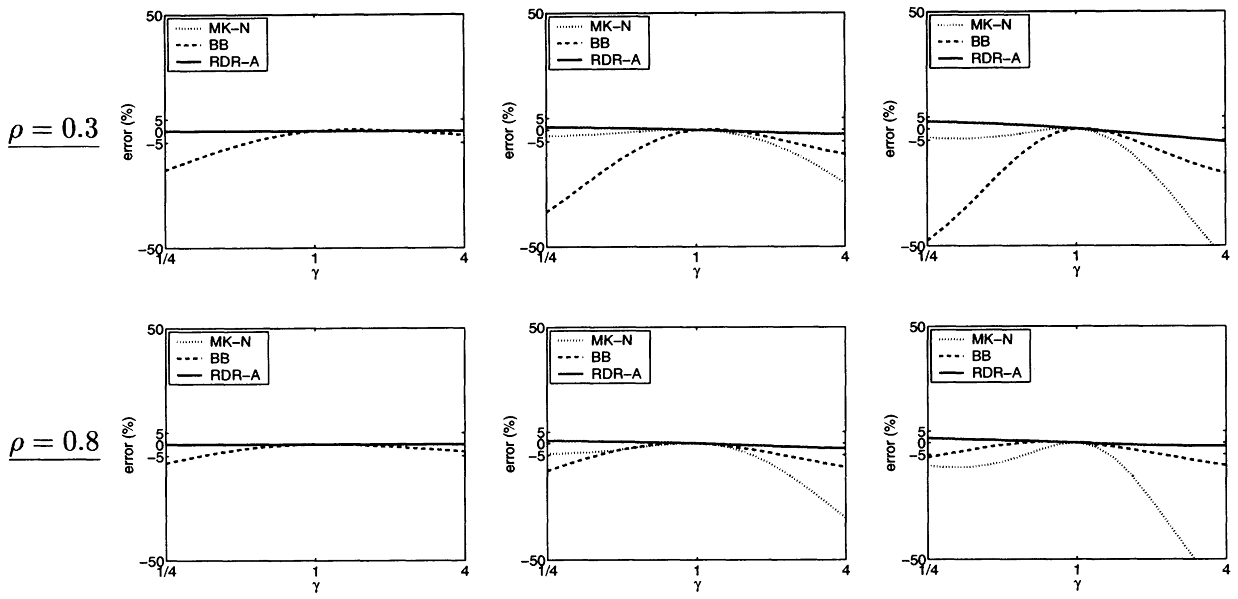
In evaluating the BB and MK-N approximations, we use the most accurate methods known to compute their components. For example, BB relies on knowing the mean delay for the M/GI/*k*/FCFS queue. We compute this delay precisely for the PH job size distribution using the matrix analytic method [18]. MK-N relies on being able to analyze the case of two priority classes (since m classes are reduced to two). We analyze the two class case in MK-N using the RDR method. We first discuss the accuracy of MK-N and RDR-A, and then discuss the accuracy of BB.

The Mitrani-King-Nishida (MK-N) and RDR-A approximations

Figure 6 shows that the error in MK-N can sometimes exceed 50%. (Note that the MK-N approximation does not show up in the bottom half of Figure 6 (PH job sizes) because the error is too great — approximating $m-1$ PH job size distributions by a single exponential clearly does not work.) By contrast, the error in RDR-A never exceeds 5% for exponential job size distributions and never exceeds 10% for PH job size distributions. We now discuss the error per class, showing that this error increases as we move towards lower-priority classes.

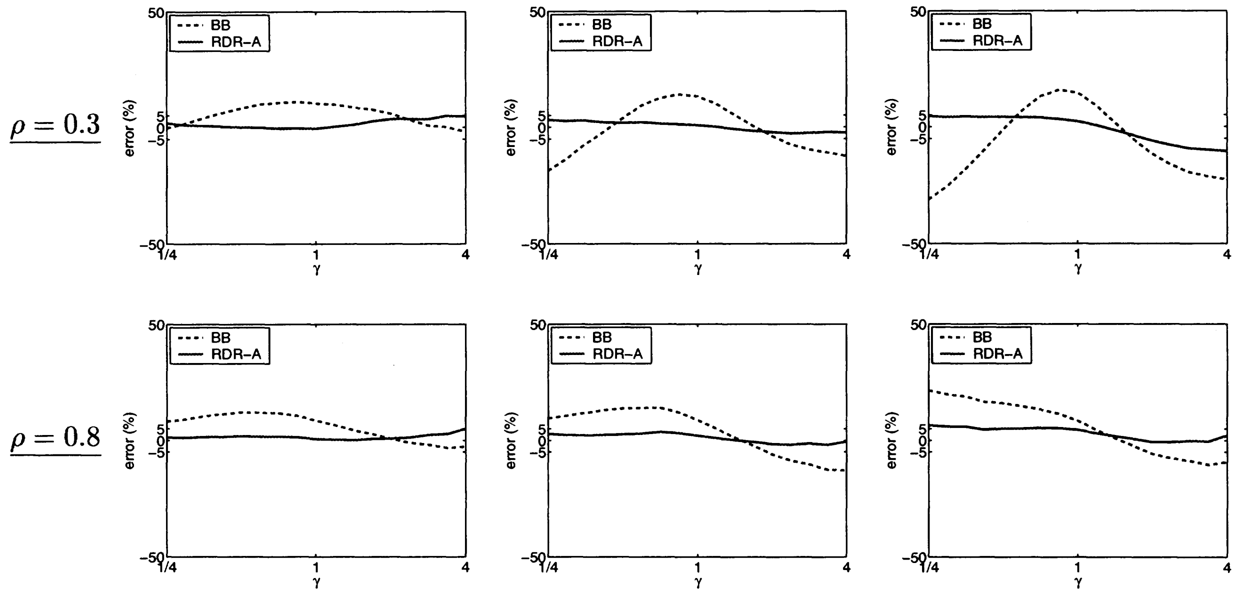
For class 2 jobs (Figure 6, column 1) the MK-N and RDR-A approximations are exact when $C^2 = 1$. This is because we use the RDR method in evaluating MK-N with two priority classes. When $C^2 = 8$, the

Comparison of approximations for M/M/k with m priority classes ($C^2 = 1$)



(a) Class 2 (b) Class 3 (c) Class 4

Comparison of approximations for M/GI/k with m priority classes ($C^2 = 8$)



(a) Class 2 (b) Class 3 (c) Class 4

Figure 6: *Comparison of RDR-A, MK-N, and BB approximations for M/GI/k with m priority classes. We consider $k = 2$ servers and $m = 4$ classes with mean $E[X_4] = 1$, $E[X_3] = \gamma$, $E[X_2] = \gamma^2$, and $E[X_1] = \gamma^3$, where the squared coefficient of variability of the job size distributions are $C^2 = 1$ (top two rows) or $C^2 = 8$ (bottom two rows) for all classes. Load is balanced among the classes. Note that MK-N does not appear for $C^2 = 8$, because the error is beyond the scale of the graphs for most values of γ.*

RDR-A approximation is still equivalent to RDR for an M/GI/k with two priority classes, and the error for RDR is always within 5% for this case.

For class 3 (column 2) error in both MK-N and RDR-A becomes apparent. Both approximations aggregate the two higher priority classes, and when $C^2 = 1$, the aggregated job size distribution is a 2 branch hyperexponential distribution. The hyperexponential distribution is approximated by an exponential distribution in the MK-N approximation, while the exact 2 branch hyperexponential distribution is used in the RDR-A approximation. Therefore, the MK-N approximation has both priority error and distribution error, while RDR-A has only priority error. We observe that the error in the RDR-A approximation (priority error only) is orders of magnitude smaller than the error in the MK-N approximation (both priority and distribution error).

The effect of γ on the error of the MK-N approximation in class 3 is significant. When $\gamma = 1$ and $C^2 = 1$, MK-N is exact, since the aggregated distribution is exponential. As γ gets smaller or larger than 1, the MK-N approximation underestimates the mean delay. This makes intuitive sense because MK-N ignores the variability among the high priority classes, which leads to a lower mean delay estimate for the lower priority jobs. We can also observe that the error in MK-N is smaller when the high priority classes are small ($\gamma < 1$) as compared to the case when $\gamma > 1$. This is because when γ is large, approximating the aggregated higher priority classes by an exponential distribution results in ignoring huge jobs in class 1, and ignoring huge jobs leads to a great underestimation in the mean delay of the low priority class.

The effect of γ on the error of RDR-A approximation in class 3 is much smaller than the effect on MK-N. When $\gamma = 1$ and $C^2 = 1$, RDR-A is exact as is MK-N. When $\gamma = 1$ and $C^2 = 8$, RDR-A does not have distribution error but does have priority error. However, as for class 2, priority error is very small in this case as well. For all ρ's and C^2's, RDR-A tends to slightly overestimates the mean delay for $\gamma < 1$, and it tends to slightly underestimates for $\gamma > 1$. This is explained by reasoning about the busy periods made up of the high priority classes. Recall that the busy period of class 1-2 jobs is defined as the period when both servers have class 1-2 jobs. When smaller jobs have priority, more larger jobs are left at the end of a busy period. Since larger jobs are less likely to be balanced between two servers, one of the servers will likely become free sooner, which allows the lower priority class to get service sooner. When larger jobs have priority, both servers are likely to become free at a similar time as low priority, smaller jobs can fill in the gaps. This causes the lower priority jobs to wait longer before they get service at either server. When $\gamma < 1$, smaller jobs have priority, but RDR-A ignores the priority and the busy period estimated by RDR-A is longer, which in turn yields overestimation of the mean delay. The underestimation for $\gamma > 1$ can be explained analogously.

Continuing with class 3, we see that the load does not significantly effect the class 3 error, but at higher load the error in MK-N is slightly larger and the error in RDR-A is slightly smaller. This implies that the priority error is smaller and the distribution error is larger at higher load. The priority error is due to the error in estimating the busy period of the higher priority jobs, and at high load the busy period is long and the relative difference between priority scheduling and FCFS with respect to the length of the busy period is smaller. Larger distribution error at higher loads can be explained through an analogy to the case of a single server. Recall equation (2), the mean delay of the lower priority class in an M/GI/1 queue with two priority classes. The distribution error in this case corresponds to the error in the second moment of the job size, $E[X^2]$, and the term with the error, $\frac{\lambda E[X^2]}{2(1-\rho_H)(1-\rho)}$, increases as the load, ρ, approaches 1.

For class 4 (column 3), we observe that the error has a similar trend as in class 3. The only difference is that the aggregation in RDR-A is no longer exact for class 4, even when $C^2 = 1$; but we still see that RDR-A always has a small error. We conclude that the priority error is small regardless of the load and the relative sizes among the classes, but ignoring the variability among the mean job sizes of higher priority classes leads to significant error.

The Buzen-Bondi (BB) approximation

We now study the accuracy of the BB approximation in Figure 6, starting at low load ($\rho = 0.3$) and exponential job size distributions (row 1). For all classes, BB is exact when all the classes have the same mean size ($\gamma = 1$), but it underestimates the mean delay when the higher priority classes are small ($\gamma < 1$) or large ($\gamma > 1$). This can be explained as follows. BB is based on the observation that the "improvement" of priority scheduling over FCFS under $k > 1$ server is similar to the case of 1 server (recall equation (1)). When all the classes have the same mean, priority scheduling provides the same mean delay as FCFS, regardless of the number of servers; hence (1) is exact, and so is BB. When high priority classes are small, priority scheduling improves the mean delay over FCFS because it allows small jobs to avoid waiting behind large jobs. However, the improvement under $k > 1$ servers is smaller than in the case of 1 server, because having $k > 1$ servers also allows small jobs to avoid waiting behind large jobs under FCFS. Since BB assumes that the improvement is the same, it underestimates the mean delay. When class 1 is larger than class 2, prioritizing class 1 worsens the mean delay. However, the penalty due to prioritization under 1 server is smaller than under $k > 1$ servers, since with 1 server FCFS also forces large jobs to block small jobs, but $k > 1$ servers often allow small jobs to avoid waiting behind large jobs under FCFS. Again, since BB assumes that the improvement is the same, it underestimates the mean delay.

For non-exponential job size distributions, BB is no longer exact even when all classes have the same job size distributions, as shown in Figure 6 rows 3-4. For $C^2 = 8$, preemptive priority scheduling and FCFS yields different mean delay. For example, for job size distributions with decreasing failure rate, low priority jobs with longer remaining time is more likely preempted by higher priority jobs, which can improve the overall mean delay. However, this improvement differs between single server systems and multiserver systems. BB therefore can yields error in predicting the mean delay.

We can also observe that the error in BB is smaller under high load than low load ($\rho = 0.3$, row 1). This is because under high load 1-server systems and k-server systems behave similarly. Overall, the error in BB can be as high as 50% under low load ($\rho = 0.3$) and 20% under high load ($\rho = 0.8$).

6 Heuristics for designing background processes

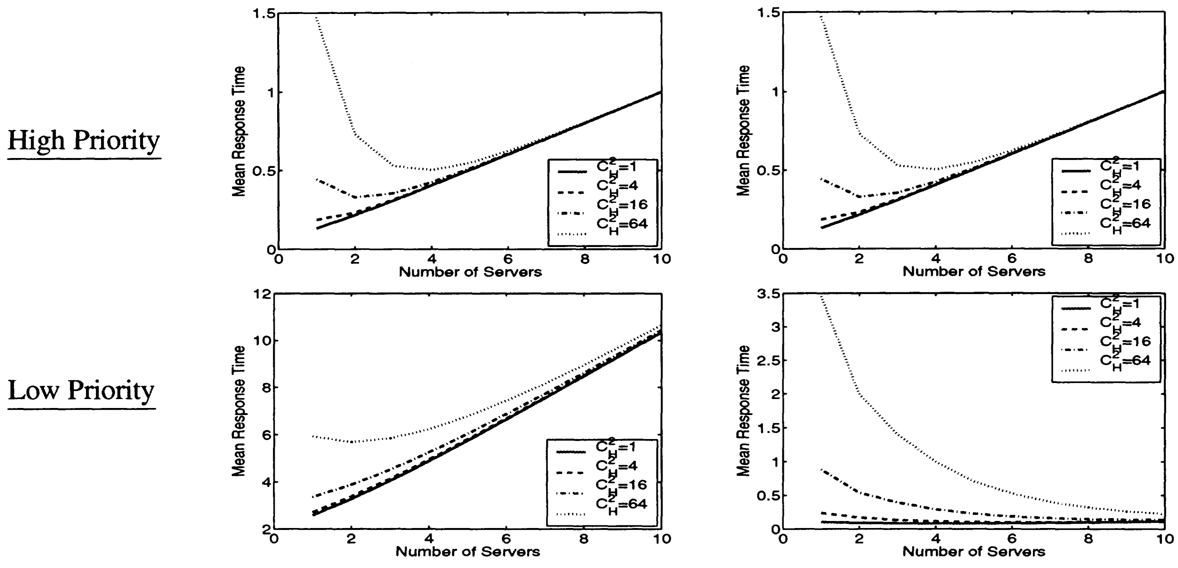
Up to this point, we have focused on validating the RDR analysis and comparing the RDR-A approximation with existing approximations for the M/GI/k priority queue. In this section, we exploit our analysis to understand how to improve the performance of modern reliable multiserver systems. In particular, modern multiprocessor systems run many reliability processes that rely on preemptive error detection, where service delivery is suspended while checks for latent errors and dormant faults are performed. These preemptions occur for the purposes of fault recovery, fault isolation, fault masking, intrusion detection, virus checking, etc.. Thus, in modern systems user-level tasks may be preempted by a wide range of higher priority, dependability tasks. We observe that there are many attributes of background processes that determine the magnitude of the effect they have on the performance of user-level tasks. By taking care in setting up these background processes, system architects can provide far superior user-level performance with little sacrifice to the reliability of the system. Our analyses of the M/GI/k priority system provides four heuristics about how to set up these background processes in order to curb the effect they have on lower priority, user-level tasks.

A first, somewhat obvious heuristic is that prioritization has a large effect on the performance of low priority jobs. Figure 4 shows that there is an order of magnitude difference in performance between class 1 and class 2 jobs, another order of magnitude difference between class 2 and class 3 jobs, and so on. In this figure, the system load is evenly balanced between all priority classes. Obviously, if the load of high priority classes can be lightened, the effect on low priority user-level processes will be lessened. Thus, one wants to limit the load made up by background processes of higher priority than user-level tasks.

A second, more surprising heuristic is that the lower priority, user-level performance is relatively unaffected by the ordering of priorities among higher priority background processes. Specifically, if there are two classes of background tasks with higher priority than user-level tasks, their relative priority is unimportant to the user. This fact is made apparent in Figure 6 which shows that the error from "priority aggregation" is small (since the total error in RDR-A is small). The insensitivity to priority ordering of higher priority classes is surprising since this priority ordering directly effects the length of the busy period in a multiserver system.

We now move to a discussion of Figure 7, which will illustrate two heuristics related to the number of servers. The performance metric in this figure is mean response time, defined as the mean delay plus the mean job size. Mean response time is a more appropriate measure for this discussion because, when we vary the number of servers while keeping the computation power constant (as in Figure 7), the mean job size varies as well.

In configuring a multiserver system, one often has a choice of using a few fast (expensive) servers, or more slow (cheap) servers, with *same total capacity*. All things being equal, one would go for the cheaper configuration. It turns out that there is also surprisingly a performance advantage in favor of the cheaper



(a) $E[X_H] = 1, E[X_L] = 10$ (b) $E[X_H] = 1, E[X_L] = \frac{1}{10}$

Figure 7: *Mean response time as a function of the number of servers, which range from 1 to 10. There are two priority classes. Column (a) shows the case where low priority jobs have a larger mean job size than high priority jobs. Column (b) shows the case where low priority jobs have a smaller mean job size than high priority jobs. The system load in these plots is* $\rho = 0.6$*, and the load is evenly balanced among the high and low priority classes.*

(many server) configuration: As shown in Figure 7, using multiple slow servers significantly curbs the effect of background processes on lower priority, user-level processes, and furthermore decreases overall mean response time. The benefit of the many slow cheap server configuration is more apparent when high priority jobs are large, as shown in Figure 7(b). In this case the multiple servers allow small low priority jobs a chance to get some service. When the high priority jobs are small, fewer servers are preferable — adding too many servers in this case is detrimental.

Finally, our analysis shows us that it is possible to structure the sizes of interrupts so as to minimize the effect of high priority background processes on lower priority user-level processes. In particular, structuring the high priority interruptions so that tasks require more deterministic computation times is much less painful to lower priority tasks than having high priority tasks that are highly variable. In Figure 7, we vary the *squared coefficient of variation*, C^2, of the high priority jobs. Figure 7 shows that variability has a significant effect on the mean response time of both high and low priority jobs, and that minimizing the variability of the background tasks is beneficial for both the background processes and the user-level processes.

7 Conclusion

We have presented Recursive Dimensionality Reduction (RDR), the first accurate analysis of the mean delay under the M/GI/k queue with m preemptive priority classes, for arbitrary k and m, where G is an arbitrary PH distribution. The key idea in RDR is to recursively make use of the analysis of the i-th class to analyze the $(i+1)$-th class, reducing an m dimensionally infinite state space into a 1 dimensionally infinite state space. The accuracy of RDR is always within 2% of simulation. RDR is computationally efficient if either the number of servers, k, or the number of classes, m, is a small constant — requiring less than 0.1 seconds per data point.

Since the computational complexity of RDR can grow exponentially when we allow both k and m to grow, we have proposed an approximation, RDR-A, that makes use of RDR for the M/GI/k queue with *two* priority classes. Although RDR-A is a straightforward extension of a known approximation by Mitrani and King, RDR-A is only made possible by making use of RDR. RDR-A is within 5% of simulation in most cases. This is very favorable compared with the error in the existing approximations of Mitrani & King (MK-N) and Buzen & Bondi (BB), which both can exhibit over 50% error.

The three approximations (RDR-A, MK-N, and BB) are studied extensively, and we have provided intuition as to when these approximations have error and why. The RDR-A and MK-N approximations aggregate higher priority classes into one, ignoring priorities among these classes, and approximating the aggregated distribution. We find that the error due to ignoring the priorities is very small. The error due to approximating the aggregated distribution is large in the case of MK-N, but can be made small (within 5% in most cases) by matching the first three moments of the distribution, as in RDR-A. The BB approximation is based on the assumption that the improvement of priority scheduling over FCFS does not depend on the number of servers. However, we find that the improvement heavily depends on the number of servers.

RDR and RDR-A have increasing importance for dependable systems both as a predictive tool and as a design tool. Security threats are an ever growing problem, resulting in the necessity of more and more security processes that make up a growing percentage of system load. Due to the huge effect that these security tasks have on user-level performance, it is important to be able to both predict their impact on response time and to design the security processes so as to minimize this impact. What we find is that these high-priority backgound tasks (e.g., intrusion detection, identity theft detection, and fault recovery) can have a tremendous impact on the user-perceived delay. For example, we have seen that if load is balanced between priority classes, each additional high priority class increases the mean delay of the priority classes below it by an order of magnitude. Under modern computer systems, which run an array of high priority background tasks to provide dependability, the delay contributed to user-level tasks can be enormous.

Fortunately, however we find that it is possible to significantly lessen these unwanted delays by carefully designing and scheduling these high-priority dependability tasks. We contribute several guidelines towards this end. We find that a key cause of delay is the variability of high priority dependability tasks. If the variability of the higher priority tasks is lessened, their impact on the lower priority user-level tasks is also greatly reduced. Another option for reducing user-level delay is to simply replace a single server system by a (cheaper) equal capacity multiserver system consisting of many slower servers. This will have two positive effects: (i) it will greatly reduce the delay induced by high priority tasks on low priority tasks, and (ii) it will reduce overall mean delay. There are also things that one does not need to worry about in designing dependable systems. For example, our analysis shows that the relative priorities among the higher priority background dependability tasks do not have much effect on user-perceived delay. Thus, system architects should not waste time figuring out the optimal prioritization, and can instead assign priorities as desired.

This paper is set in the context of dependable multiserver systems; however multiserver priority queues have much broader applicability, particularly in the context of Quality of Service. For any service that is desired by many users (e.g. retail Web sites, supercomputing centers, online data warehouses), it is common to utilize server farms to handle the great load. In these settings, customers are frequently willing to pay more for lower mean delays. The typical mechanism for providing reduced delay is to prioritize jobs. Customers pay to receive a certain priority level for their jobs. However, it is entirely non-obvious what that priority buys you in a multiserver system. If a customer pays for priority 2 class service, how much lower will his delay be than if he only purchases priority 3 class service? The analysis in this paper allows us to accurately answer this question for the first time, and to give accurate mean performance guarantees to customers.

References

[1] A. Avizienis, J. C. Laprie, and B. Randell. Fundamental concepts of dependability. Technical report, Research Report N01145, LAAS-CNRS, 2001.

[2] K. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36:37–53, 1993.

[3] A. Bondi and J. Buzen. The response times of priority classes under preemptive resume in M/G/m queues. In *ACM Sigmetrics*, pages 195–201, August 1984.

[4] J. Buzen and A. Bondi. The response times of priority classes under preemptive resume in M/M/m queues. *Operations Research*, 31:456–465, 1983.

[5] J. F. Chiu and G. M. Chiu. Placing forced checkpoints in distributed real-time embedded systems. *Computing & Control Engineering Journal*, 13:197–205, 2002.

[6] I. Cidon and M. Sidi. Recursive computation of steady-state probabilities in priority queues. *Operations Research Letters*, 9:249–256, 1990.

[7] W. Feng, M. Kawada, and K. Adachi. Analysis of a multiserver queue with two priority classes and (M,N)-threshold service schedule ii: preemptive priority. *Asia-Pacific Journal of Operations Research*, 18:101–124, 2001.

[8] H. Gail, S. Hantler, and B. Taylor. Analysis of a non-preemptive priority multiserver queue. *Advances in Applied Probability*, 20:852–879, 1988.

[9] H. Gail, S. Hantler, and B. Taylor. On a preemptive Markovian queues with multiple servers and two priority classes. *Mathematics of Operations Research*, 17:365–391, 1992.

[10] M. Harchol-Balter, C. Li, T. Osogami, A. Scheller-Wolf, and M. Squillante. Task assignment with cycle stealing under central queue. In *International Conference on Distributed Computing Systems*,

pages 628–637, 2003.

[11] E. Kao and K. Narayanan. Computing steady-state probabilities of a nonpreemptive priority multi-server queue. *Journal on Computing*, 2:211–218, 1990.

[12] E. Kao and K. Narayanan. Modeling a multiprocessor system with preemptive priorities. *Management Science*, 2:185–97, 1991.

[13] E. Kao and S. Wilson. Analysis of nonpreemptive priority queues with multiple servers and two priority classes. *European Journal of Operational Research*, 118:181–193, 1999.

[14] A. Kapadia, M. Kazumi, and A. Mitchell. Analysis of a finite capacity nonpreemptive priority queue. *Computers and Operations Research*, 11:337–343, 1984.

[15] H. Kopetz and R. Obermaisser. Temporal composability. *Computing & Control Engineering Journal*, 13:156–162, 2002.

[16] C. M. Krishna and A. D. Singh. Reliability of checkpointed real-time systems using time redundancy. *IEEE Transactions on Reliability*, 42:427–435, 1993.

[17] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21:558–565, 1978.

[18] G. Latouche and V. Ramaswami. *Introduction to Matrix Analytic Methods in Stochastic Modeling*. ASA-SIAM, Philadelphia, 1999.

[19] D. Miller. Steady-state algorithmic analysis of M/M/c two-priority queues with heterogeneous servers. In R. L. Disney and T. J. Ott, editors, *Applied probability - Computer science, The Interface, volume II*, pages 207–222. Birkhauser, 1992.

[20] I. Mitrani and P. King. Multiprocessor systems with preemptive priorities. *Performance Evaluation*, 1:118–125, 1981.

[21] M. Neuts. Moment formulas for the Markov renewal branching process. *Advances in Applied Probabilities*, 8:690–711, 1978.

[22] B. Ngo and H. Lee. Analysis of a pre-emptive priority M/M/c model with two types of customers and restriction. *Electronics Letters*, 26:1190–1192, 1990.

[23] T. Nishida. Approximate analysis for heterogeneous multiprocessor systems with priority jobs. *Performance Evaluation*, 15:77–88, 1992.

[24] T. Osogami and M. Harchol-Balter. A closed-form solution for mapping general distributions to minimal PH distributions. In *Performance TOOLS*, pages 200–217, 2003.

[25] T. Osogami, M. Harchol-Balter, and A. Scheller-Wolf. Analysis of cycle stealing with switching cost. In *ACM Sigmetrics 2003*, pages 184–195, 2003.

[26] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27:228–234, 1980.

[27] A. Romanovsky, J. Xu, and B. Randell. Exception handling in object-oriented real-time distributed systems. In *Symposium on Object-oriented Real-time distributed computing*, pages 32–42, 1998.

[28] A. Scheller-Wolf. Necessary and sufficient conditions for delay moments in FIFO multiserver queues with an application comparing s slow servers with one fast one. *Operations Research*, 51:748–758, 2003.

[29] A. Sleptchenko. Multi-class, multi-server queues with non-preemptive priorities. Technical Report 2003-016, EURANDOM, Eindhoven University of Technology, 2003.

[30] A. Sleptchenko, A. van Harten, and M. van der Heijden. An exact solution for the state probabilities of the multi-class, multi-server queue with preemptive priorities, 2003 – Manuscript.

[31] W. Whitt. The impact of a heavy-tailed service-time distribution upon the M/GI/s waiting-time distribution. *Queueing Systems*, 36:71–87, 2000.

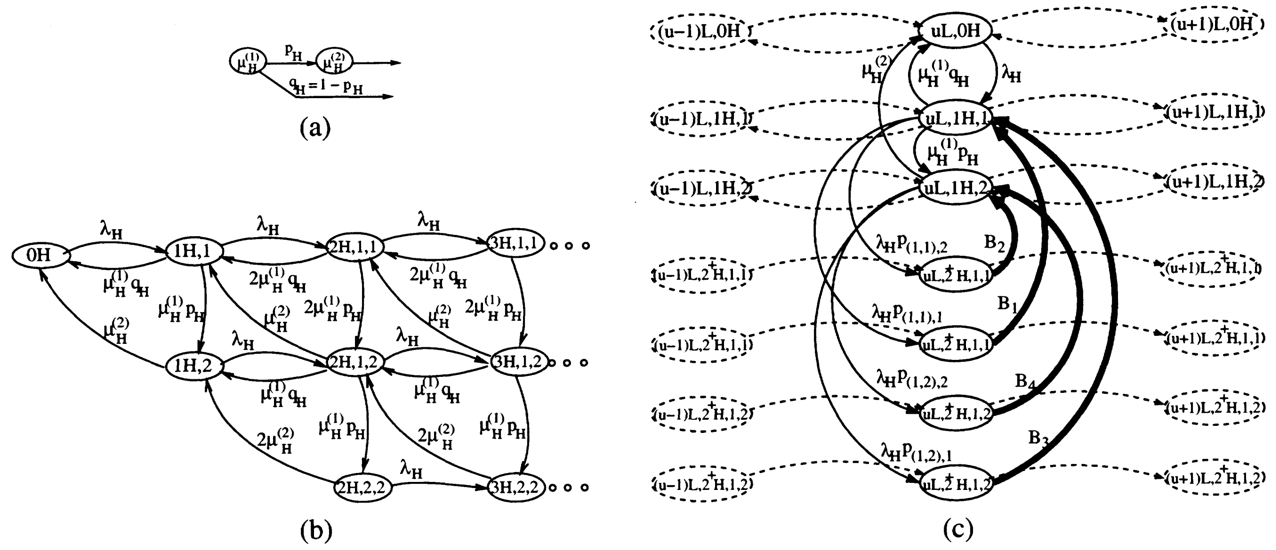


Figure 8: *(a) A 2-phase PH distribution with Coxian representation. (b) Markov chain which will be used to compute the high-priority job busy periods, in the case where high-priority job size have a PH distribution with Coxian representation shown in (a). (c) Markov chain for system with 2 servers and 2 priority classes where high priority jobs have Coxian service times.*

A Analysis of M/PH/k with m priority classes

In this section, we describe how RDR can be applied to analyze the case of PH job size distributions. We describe RDR for the case of two servers ($k = 2$) and two priority classes ($m = 2$), but this can be easily generalized to higher k's and higher m's. (To generalize to $m > 2$ classes and PH distributions, the recursive algorithm introduced in Section 3.2 can again be applied.)

The mean delay of high priority jobs can be analyzed as the mean delay in an M/PH/2 queue independently of the low priority jobs. In our description, we assume that the job size has a particular 2-phase PH distribution with Coxian representation, shown in Figure 8 (a). Under this job size distribution, a job starts in phase 1 where it is processed for a time exponentially distributed with rate $\mu_H^{(1)}$, and then either completes (with probability $q_H = 1 - p_H$) or moves to phase 2 (with probability p_H). We can then define a Markov chain as shown in Figure 8 (b), where states track the number of high priority jobs in the system and the phases of the jobs being processed. Namely, at state (0H) there are no high priority jobs in the system; at state (1H,i) there is one high priority job in the system and a job is being processed and in phase i for $i = 1, 2$; at state (nH,i, j) there are n high priority jobs in the system and two jobs are being processed and are in phase i and j, respectively (jobs in queue are all in phase 1) for $(i, j) = (1, 1), (1, 2), (2, 2)$. The limiting probabilities in this chain can be solved via the matrix analytic method, and this in turn gives the mean delay via Little's law.

The mean delay of low priority jobs can be analyzed via RDR once the busy periods of the high priority jobs are analyzed. Observe, however, that there are four different types of busy periods of high priority jobs, depending on the phases of the two jobs starting the busy period (1 & 1 or 1 & 2) and the phase of the job left at the end of the busy period (1 or 2). (Note that a busy period can never start with two jobs both in phase 2.) Since these busy periods correspond to the first passage time from a state in level l to a state in level $l-1$ in the Markov chain shown in Figure 8 (b), Neuts' algorithm can be applied to analyze these busy periods.

Figure 8 (c) shows a level of the Markov chain that tracks the number of low priority jobs, where the number of low priority jobs is u. The low priority jobs are assumed to be exponentially distributed, but this can be generalized to PH distributions. In state (uL,0H), no high priority jobs are in system. An arrival of a high priority job in state (uL,0H) triggers a transition to state (uL,1H,1). In state (iL,1H,j), one high priority job in phase j is in the system for $j = 1, 2$. An arrival of a high priority job in state (uL,1H,j) triggers a transition to state (iL,2$^+$H,1,j) for $j = 1, 2$. In state (iL,2$^+$H,1,j), at least two high priority jobs are in the system, and the two jobs that started the busy period were in phase j and k, respectively, for $j = 1, 2$ and $k = 1, 2$. The four types of busy periods are labeled as B_1, B_2, B_3, and B_4, and the duration of these busy periods is approximated by PH distributions by matching first three moments of the distribution. Finally, $p_{(i,j),k}$ denotes the probability that a busy period started by two jobs in phase i and j ends with a job in phase k left in the system for $i = 1, 2$, $j = 1, 2$, and $k = 1, 2$. This analysis is validated against simulation in Section 4.

B Moments of busy periods in QBD processes

Neuts' algorithm [21] is an efficient algorithm that calculates the moments of various types of busy periods in very general processes, i.e. M/G/1 type semi-Markov processes. Because of its generality, however, the description of the algorithm in [21] is sophisticated, and thus non-trivial to understand or implement. Since Neuts' algorithm can be applied to the performance analysis of many computer and communication systems, it is a shame that it has not been used more frequently in the literature.

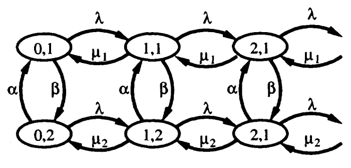
The purpose of this section is therefore to make Neuts' algorithm more accessible by re-describing his algorithm restricted to the first three moments of particular types of busy periods in QBD processes. We omit all proofs, which are provided in detail in [21]; instead we focus on intuition and interpretation. We include everything needed to apply Neuts' algorithm within our solution framework, so that readers who wish to apply our methodology can do so.

In our paper, we consider a QBD process with state space $E = \{(i,j) | i \geq 0, 1 \leq j \leq m\}$, which has generator matrix Q:

$$Q = \begin{pmatrix} B & A_0 & 0 & \cdots \\ A_2 & A_1 & A_0 & \vdots \\ 0 & A_2 & A_1 & \ddots \\ \vdots & \cdots & \ddots & \ddots \end{pmatrix}$$

where B and A_i are $m \times m$ matrices. Figure 9 shows a particular QBD process with $m = 2$.

We define level i as denoting the set of states of the form (i, j) for $j = 1, ..., m$. Our goal can be roughly stated as deriving the passage time required to get from state $(1, j)$ to level 0 conditioned on the particular state first reached in level 0. More precisely, we seek the first three moments of the distribution of the time required to get from state $(1, j)$ to state $(0, k)$, given that state $(0, k)$ is the first state reached in level 0. In the rest of this section, we describe how Neuts' algorithm can be applied to derive these quantities.



(a)

$$
\begin{aligned}
B &= \begin{pmatrix} -(\lambda+\alpha) & \alpha \\ \beta & -(\lambda+\beta)\lambda \end{pmatrix} \\
A_0 &= \begin{pmatrix} \lambda & 0 \\ 0 & \lambda \end{pmatrix} \\
A_1 &= \begin{pmatrix} -(\lambda+\mu_1+\alpha) & \alpha \\ \beta & -(\lambda+\mu_1+\beta)\lambda \end{pmatrix} \\
A_2 &= \begin{pmatrix} \mu_1 & 0 \\ 0 & \mu_2 \end{pmatrix}
\end{aligned}
$$

(b)

Figure 9: *An example of a QBD process: (a) states and transition rates, and (b) submatrices of the generator matrix.*

Notation

We define the transition probability matrix, $P(x)$, as

$$
P(x) = \begin{pmatrix} \beta(x) & \alpha_0(x) & 0 & \cdots \\ \alpha_2(x) & \alpha_1(x) & \alpha_0(x) & \vdots \\ 0 & \alpha_2(x) & \alpha_1(x) & \ddots \\ \vdots & \cdots & \ddots & \ddots \end{pmatrix}
$$

where the (s,t) element, $P_{st}(x)$, is the probability that the sojourn time at state (i_s, j_s) is less than or equal to x and the first transition out of state (i_s, j_s) is to state (i_t, j_t). Note that a state is a pair of numbers, level and phase, and formally $i_s m + j_s = s$, $i_t m + j_t = t$, and $j_s, j_t \leq m$ hold. Observe also that $\beta(x)$ and $\alpha_i(x)$ are each $m \times m$ submatrices for $i = 0, 1, 2$.

Next, we define the r-th moment of submatrices $\alpha_i(x)$ as $\alpha_i^{(r)} = \int_0^\infty x^r d\alpha_i(x)$ for $i = 0, 1, 2$ and $r = 1, 2, 3$, where an integral of a matrix M is a matrix of the integrals of the elements in M.

Example

Consider the QBD process shown in Figure 9. Let $\gamma_1 = \lambda + \mu_1 + \alpha$ and $\gamma_2 = \lambda + \mu_2 + \beta$. Then, $\alpha_i(x)$'s and their moments $\alpha_i^{(r)}$ for $r = 1, 2, 3$ look as follows:

$$
\begin{aligned}
\alpha_0(x) &= \begin{pmatrix} 1-e^{-\gamma_1 x} & 0 \\ 0 & 1-e^{-\gamma_2 x} \end{pmatrix} \begin{pmatrix} \frac{\lambda}{\gamma_1} & 0 \\ 0 & \frac{\lambda}{\gamma_2} \end{pmatrix} \\
\alpha_1(x) &= \begin{pmatrix} 1-e^{-\gamma_1 x} & 0 \\ 0 & 1-e^{-\gamma_2 x} \end{pmatrix} \begin{pmatrix} 0 & \frac{\alpha}{\gamma_1} \\ \frac{\beta}{\gamma_2} & 0 \end{pmatrix} \\
\alpha_2(x) &= \begin{pmatrix} 1-e^{-\gamma_1 x} & 0 \\ 0 & 1-e^{-\gamma_2 x} \end{pmatrix} \begin{pmatrix} \frac{\mu_1}{\gamma_1} & 0 \\ 0 & \frac{\mu_2}{\gamma_2} \end{pmatrix}
\end{aligned}
$$

and

$$
\alpha_0^{(r)} = \begin{pmatrix} \frac{r!}{\gamma_1^r} & 0 \\ 0 & \frac{r!}{\gamma_2^r} \end{pmatrix} \begin{pmatrix} \frac{\lambda}{\gamma_1} & 0 \\ 0 & \frac{\lambda}{\gamma_2} \end{pmatrix}
$$

$$\begin{aligned}
\alpha_1^{(r)} &= \begin{pmatrix} \frac{r!}{\gamma_1^r} & 0 \\ 0 & \frac{r!}{\gamma_2^r} \end{pmatrix} \begin{pmatrix} 0 & \frac{\alpha}{\gamma_1} \\ \frac{\beta}{\gamma_2} & 0 \end{pmatrix} \\
\alpha_2^{(r)} &= \begin{pmatrix} \frac{r!}{\gamma_1^r} & 0 \\ 0 & \frac{r!}{\gamma_2^r} \end{pmatrix} \begin{pmatrix} \frac{\mu_1}{\gamma_1} & 0 \\ 0 & \frac{\mu_2}{\gamma_2} \end{pmatrix}
\end{aligned}$$

Finally, let $G(x)$ be an $m \times m$ matrix whose (j,k) element, $G_{jk}(x)$, is the probability that the time to visit level 0 is at most x and the first state visited in level 0 is $(0,k)$ given that we started at $(1,j)$. Also, let $G_{jk}^{(r)}$ be the r-th moment of $G_{jk}(x)$; namely, $G^{(r)} = \int_0^\infty x^r dG(x)$ for $r = 1,2,3$.

Matrix $G = \lim_{x\to\infty} G(x)$ is a fundamental matrix used in the matrix analytic method, and many algorithms to calculate G have been proposed [18]. The most straightforward (but slow) algorithm is to iterate

$$G = -A_1^{-1}A_0GG - A_1^{-1}A_2 \tag{3}$$

until it converges. Notice that $G_{jk}(x)$ is not a proper distribution function and $G_{jk} = \lim_{x\to\infty} G(x)$, which is the probability that the first state in level 0 is state $(0,k)$ given that we start at state $(1,j)$, can be less than 1. Therefore, $G_{jk}^{(r)}$ is not a proper moment rather a conditional moment: "the r-th moment of the distribution of the first passage time to level 0 given that the first state in level 0 is state $(0,k)$ and given that we start at state $(1,j)$" multiplied by "the probability that the first state reached in level 0 is state $(0,k)$ given that we start at state $(1,j)$."

Moments of busy periods

The quantities that we need in our methodology are (a) the probability that the first state reached in level 0 is state $(0,k)$ given that we start at state $(1,j)$ and (b) the r-th moment of the distribution of the first passage time to level 0 given that the first state in level 0 is state $(0,k)$ and given that we start at state $(1,j)$. Quantity (a) is given by G_{jk}, and quantity (b) is given by $\frac{G_{jk}^{(r)}}{G_{jk}}$. Therefore, our goal is to derive matrices, G and $G^{(r)}$ for $r = 1,2,3$.

Matrix G is obtained by an iterative substitution of (3). Once G is obtained, matrix $G^{(1)}$ is obtained by iterating

$$G^{(1)} = \alpha_2^{(1)} + \alpha_1^{(1)}G + \alpha_1 G^{(1)} + \alpha_0^{(1)}GG + \alpha_0 G^{(1)}G + \alpha_0 GG^{(1)} \tag{4}$$

Similarly, matrix $G^{(2)}$ is obtained by iterating

$$\begin{aligned}
G^{(2)} &= \alpha_2^{(2)} + \alpha_1^{(2)}G + 2\alpha_1^{(1)}G^{(1)} + \alpha_1 G^{(2)} \\
&\quad + \alpha_0^{(2)}GG + 2\alpha_0^{(1)}(G^{(1)}G + GG^{(1)}) + \alpha_0(G^{(2)}G + 2G^{(1)}G^{(1)} + GG^{(2)})
\end{aligned} \tag{5}$$

and matrix $G^{(3)}$ is obtained by iterating

$$\begin{aligned}
G^{(3)} &= \alpha_2^{(3)} + \alpha_1^{(3)}G + 3\alpha_1^{(2)}G^{(1)} + 3\alpha_1^{(1)}G^{(2)} + \alpha_1 G^{(3)} \\
&\quad + \alpha_0^{(3)}GG + 3\alpha_0^{(2)}(G^{(1)}G + GG^{(1)}) + 3\alpha_0^{(1)}(G^{(2)}G + 2G^{(1)}G^{(1)} + GG^{(2)}) \\
&\quad + \alpha_0(G^{(3)}G + 3G^{(2)}G^{(1)} + 3G^{(1)}G^{(2)} + GG^{(3)}).
\end{aligned} \tag{6}$$

We now give intuition behind expressions (4), (5), and (6). The right hand side of (4) can be divided into three parts: [0] $\alpha_2^{(1)}$, [1] $\alpha_1^{(1)}G + \alpha_1 G^{(1)}$, and [2] $\alpha_0^{(1)}GG + \alpha_0 G^{(1)}G + \alpha_0 GG^{(1)}$. For $h = 0,1,2$, the (j,k) element of part $[h]$ gives "the first moment of the distribution of the time t required to get from state $(1,j)$ to state $(0,k)$ given that the first transition out of state $(1,j)$ is to level h and the first state reached in level 0 is

$(0, k)$" multiplied by "the probability that the first transition out of state $(1, j)$ is to level h and the first state reached in level 0 is $(0, k)$." Part [1] consists of two terms. The first term, $\alpha_1^{(1)}G$, is the contribution to t made up of the time until the first transition, and the second term, $\alpha_1 G^{(1)}$, is the contribution to t from when the first transition occurs until we reach $(0, k)$. Similarly, part [2] consists of three terms. The first term, $\alpha_0^{(1)}GG$, is the contribution to t made up by the time until the first transition; the second term, $\alpha_0 G^{(1)}G$, is the contribution to t from consisting of the time to return to level 1 from level 2; finally, the third term, $\alpha_0 GG^{(1)}$, is the contribution to t of the time required to go from level 2 to level 1.

The right hand sides of (5) and (6) can similarly be divided into three parts: part [0] consists of terms containing α_2 or $\alpha_2^{(r)}$; part [1] consists of terms containing α_1 or $\alpha_1^{(r)}$; part [2] consists of terms containing α_0 or $\alpha_0^{(r)}$. The three parts of (5) and (6) can be interpreted exactly the same way as the three parts of (4) except that "the first moment" in (4) must be replaced by "the second moment" and "the third moment" in (5) and (6), respectively. The three terms in part [1] of (5) can be interpreted as follows. Let T_α be the time to the first transition and let T_G be the time it takes from level 1 to level 0. Then, the second moment of the distribution of these two times is

$$E[(T_\alpha + T_G)^2] = E[(T_\alpha)^2] + 2E[T_\alpha]E[T_G] + E[(T_G)^2],$$

since T_α and T_G are independent. Roughly speaking, $\alpha_1^{(2)}G$ corresponds to $E[(T_\alpha)^2]$, $2\alpha_1^{(1)}G^{(1)}$ corresponds to $2E[T_\alpha]E[T_G]$, and $\alpha_1 G^{(2)}$ corresponds to $E[(T_G)^2]$. The other terms can be interpreted in the same way.

Generalizations

Finally, we mention four generalizations that Neuts' algorithm allows. First, while we restricted ourselves to the first three moments, Neuts' method generalizes to any higher moments. Second, while we restricted ourselves to the first passage time from level 1 to level 0, Neuts' method generalizes to passage times from level i from level 0. Third, while we limited our discussion to QBD processes, the results can be generalized to M/G/1 type semi-Markov processes. Fourth, while we only computed moments of the distribution of the duration of busy periods, one can also compute moments of the joint distribution of the duration of a busy period and the number of transitions during the busy period. These four generalizations are all formally proven in [21].