How many servers are best in a dual-priority FCFS system?

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

November 2003
CMU-CS-03-201

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We ask the question, "for minimizing mean response time, which is preferable: one fast server of speed 1, or k slow servers each of speed $1/k$?" Our setting is the M/GI/k system with two priority classes of customers, high priority and low priority, where G is a phase-type distribution. We find that multiple slow servers are often preferable — and we demonstrate exactly how many servers are preferable as a function of load and G. In addition, we find that the optimal number of servers with respect to the high priority jobs may be very different from that preferred by low priority jobs, and we characterize these preferences. We also evaluate the optimal number of servers with respect to overall mean response time, averaged over high and low priority jobs. Lastly, we ascertain the effect of the variability of high priority jobs on low priority jobs.

This paper is the first to analyze an M/GI/k system with two priority classes and a general phase-type distribution. Prior analyses of the M/GI/k with two priority classes either require that G be exponential, or are approximations that work well when G is exponential, but are less reliable for more variable G. Our analytical method is very different from the prior literature: it combines the technique of dimensionality reduction (see [9]) with Neuts' technique for determining busy periods in multiserver systems [22]. Our analysis is approximate, but can be made as accurate as desired, and is verified via simulation.

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

This work was supported by NSF Grant CCR-0311383, and grant sponsorship from IBM Corporation.

Keywords: Scheduling, queueing, multiserver, priority, preemptive, FCFS, response time, M/GI/k, dimensionality reduction, busy period, phase type.

1 Introduction

In a world filled with competing retailers, online stores and service providers are forced to worry about response times experienced by customers.[1] In an effort to maximize profit, it is becoming commonplace to prioritize online service in favor of customers who are "big spenders" (either currently or in recent history), providing shorter response times for those customers at the expense of low-priority customers. Not only is prioritization of customers common at web servers where the motivation is keeping the big spenders happy [17], it is also common at supercomputing centers where certain customers pay for better service and at airlines where customers can pay for first class service. A job may also be given high priority even when not associated with the purchase of a high-cost item, simply because the job is important in keeping the system functioning property, such as certain operating system function calls or security procedures. In these examples, customers within the *same* class are often served in first-come-first-serve (FCFS) order for fairness.

For such systems with dual priorities (high-priority jobs and low-priority jobs), it is important to understand the mean response time experienced by each class of jobs, as well as the overall mean response time averaged over both priority classes. Further complicating analysis, today's systems often involve multiple servers in a server farm configuration, as it is cheaper to purchase k machines each of speed $1/k$ than to purchase one fast machine of speed 1. While the response time for a single server M/GI/1 with dual priorities is well-understood, the mean response time of a k-server M/GI/k system with dual priorities is *not known*; only approximations exist for all but the case of exponentially-distributed service times for both priority classes. (These are summarized in Section 2).

The first contribution of this paper is thus an analysis of the M/GI/k with dual preemptive-resume priorities, where each priority class has job sizes (job service requirements) following a general phase-type (PH) distribution (see Figure 1), and jobs within the same class are served in FCFS order. Observe that the set of PH distributions is very general and is provably dense in the set of all non-negative distributions [10]. In this general setting, we derive the mean response time for each priority class as well as the overall mean response time averaged over both classes. Our analysis combines two techniques. The first is the method of Dimensionality Reduction (see [9]), which allows us to reduce a 2-dimensionally-infinite Markov chain tracking the number of jobs of each priority class to a 1-dimensionally-infinite Markov chain. The second technique, introduced by Neuts in [22], allows the computation of the complex busy periods used within the Dimensionality Reduction method. While our method is inherently approximate, in that we match a finite number of moments of busy periods, it can be made as accurate as desired. Our validation against simulation shows that our analysis is always within 1% accuracy using just three moments.

Our results for the M/GI/k with dual priorities show that the response times of both priority classes are strongly affected by the number of servers, k (each serving at rate $1/k$), and this effect is sometimes counter-intuitive. This question of how the number of servers (with fixed total processing power 1) can affect mean response time has never been investigated in the case of a dual-class system. Even for a single-class system, this issue has only been looked at only as the number of servers goes to infinity, or for particular classes of job size distributions (see Section 2).

[1] Here response time refers to the total time from when a client issues a transaction/purchase until the purchase is complete – this includes delay and service time.

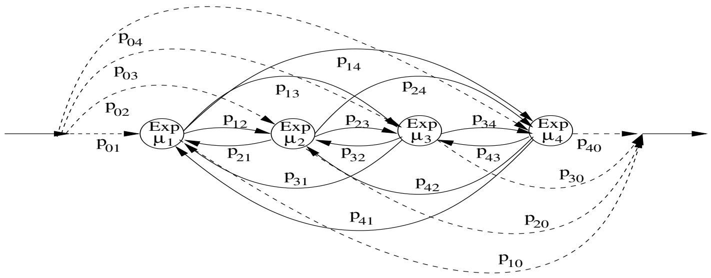


Figure 1: A PH distribution is the distribution of the absorption time in a finite state continuous time Markov chain. The figure shows a 4-phase PH distribution, with $n = 4$ states, where the ith state has exponentially-distributed sojourn time with rate μ_i. With probability p_{0i} we start in the 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.

The second contribution of this paper is thus to provide a better understanding of response times under multiserver priority systems. In particular, we ask the following three high-level questions in the context of an M/GI/k dual-priority queue with preemptive-resume:

1. Are multiple cheap slow servers ever better than a single expensive fast server?

2. Does the answer to "how many servers are optimal" differ for the different priority classes? Also, does the overall optimal number of servers in a dual-priority system differ from the case where the classes are indistinguishable — that is when all jobs have been aggregated into a single priority class?

3. How does the number of servers affect the overall mean response time, as well as that of low and high priority jobs?

We find that the answers to these questions depend on the relative sizes and relative proportions (loads) of the classes, as well as the variability[2] of high priority jobs. In particular, in answer to the first question, we find that many cheap slow servers can in fact be better than a single expensive fast server: as the variability of the high priority jobs is increased and/or the overall load is increased, the number of servers preferred greatly increases. In fact, we find

[2] Throughout the paper the variability of a random variable X will be expressed in terms of $C^2 = E[X^2]/E[X]^2 - 1$. The system load, ρ, will be defined as $\lambda E[X]/k$, where λ is the arrival rate, X represents job size, and k is the number of servers.

the response times experienced under many slow servers compared to that under a single fast server often differ substantially.

In answer to the second question, the number of servers preferred by low priority versus high priority jobs can vary widely. For example, when high priority jobs are much smaller than low priority jobs, low priority jobs prefer far fewer servers than do high priority jobs. By contrast, when low priority jobs are smaller than high priority jobs, the low priority jobs prefer far more servers than the high priority jobs. We also find that the overall optimal number of servers in the dual priority case, averaged over both classes, can differ substantially from the optimal number of servers in the single class aggregate case.

In answer to the third question, increasing the number of servers (while fixing the service capacity) can be beneficial up to a point, providing significant improvement in overall mean response time. However, beyond a point, increasing the number of servers is severely detrimental to the overall system performance. Thus, when viewed as a function of the number of servers in the system, the overall mean response time curve forms a "U-shape" where the bottom of the "U" corresponds to the optimal number of servers.

Detailed explanations of these results and many others are given in Section 6. It is our hope that system designers will be able to use the results in this paper both as a predictive and design tool, to predict performance in dual-priority server farms and also to design server farms with improved response times, sometimes at a substantially reduced cost.

2 Prior work

The ubiquity of multi-server (M/GI/k) systems has given rise to much research on their performance. Included in this research are a number of papers on dual priority systems, where high priority jobs have preference over low priority jobs. In such research the question of interest is typically deriving the mean response time for each priority class. Throughout the discussion below, that question will be central.

2.1 Dual priority analysis in an M/GI/k

Almost all the papers written on dual priority M/GI/k systems are restricted to exponential service times (Section 2.1.1). The few papers which deal with non-exponential service time are either coarse approximations or only handle limited settings (Section 2.1.2). Our paper differs from all of the prior work in that we can solve dual-priority systems for an M/GI/k to an arbitrary level of accuracy, where G is any *phase-type* (PH) distribution.

2.1.1 Exponential service times

Techniques for analyzing the M/M/k dual priority system can be organized into four types on which we elaborate below: (i) approximations via aggregation or truncation; (ii) matrix analytic methods; (iii) generating function methods; (iv) special cases where the priority classes have the same mean. Unless otherwise mentioned, preemptive-resume priorities should be assumed.

Nearly all analysis of dual priority multiserver systems involves the use of a Markov chain. Unfortunately, even with two priority classes, the state space of such a chain grows infinitely in two dimensions, one dimension for each priority class. In order to overcome this, researchers have simplified the chain in various ways. Kao and Narayanan truncate the chain by either limiting the number of high priority jobs [11], or the number of low priority jobs [12]. Nishida aggregates states, yielding an often rough approximation [24]. Kapadia, Kazmi and Mitchell explicitly model a finite queue system [14]. Unfortunately, aggregation or truncation is unable, in general, to capture the system performance as the traffic intensity grows large.

Matrix analytic methods also are *unable to handle two-dimensionally infinite systems*. In order to overcome this, Miller [18] and Ngo and Lee [23] partition the state space into blocks and then "super-blocks," according to the number of high priority customers in queue. This partitioning is quite complex and is unlikely to be generalizable to non-exponential job sizes. In addition, [18] experiences numerical instability issues when $\rho > 0.8$.

A third stream of research capitalizes on the exponential job sizes by explicitly writing out the system balance equations and then finding roots via generating functions. These techniques in general yield complicated mathematical expressions (susceptible to numerical instabilities at higher loads). King and Mitrani use this technique, yielding exact values for a two-class system and some very rough approximations for larger numbers of classes [19]. For larger systems (eight servers) they begin to experience numerical stability problems – their solution yields some negative probabilities. Gail, Hantler, and Taylor [7, 8] follow a similar approach and also report stability problems. Feng, Kowada, and Adachi [6] generalize priority service to more general switching thresholds. Kao and Wilson look at the non-preemptive case by using the power series method, with refinements to help ensure convergence [13]. They then compare the performance of this algorithm to three matrix analytic algorithms. The power series method takes much less time than its matrix analytical counterparts, but does at times show numerical problems – significant deviations from the correct value established independently.

Finally there are papers that consider the special case where the multiple priority classes all having the same mean. These include Davis [5], and Buzen and Bondi [2].

2.1.2 General service times

In a follow-up to their 1983 paper, Bondi and Buzen extend their technique to general service distributions and multiple classes [1], by starting with the M/GI/k/FCFS queue and multiplying its delay by a scaling factor based on a single server. As no exact results exist, they compare their approximations against simulation. In order to understand how our results compare with those in [1], we will plot our results against their approximations directly in Section 5, and show that our techniques yield much more accurate results.

The only work dealing with non-exponential service times in a more precise setting is a pair of papers, not yet published, by Sleptchenko [27] and Sleptchenko et. al. [28]. Both papers consider a two-priority, multiserver system where within each priority there may be a number of different classes, 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 generalize to distributions other than the hyperexponential [27, 28].

2.2 How many servers are best in an M/GI/k?

The question of how many servers are best has never been addressed in the context of a dual priority system. For single-priority systems, however, the question of how many servers are best has a long history in the literature. In describing the results below, and throughout the paper, we will assume that the performance metric of interest is mean response time rather than mean queueing time, since it's obvious that to minimize mean queueing time one wants an infinite number of servers [3, 4]. The results below all deal with only a single priority class.

As early as 1958 Morse observed that for an M/M/k system the optimal number of servers is one [21]. This was formalized by Stidham [29], who showed that under a general arrival process, and service times that are exponential, Erlang, or deterministic, a single server minimizes the expected number in system.

Mandelbaum and Reiman [16] prove that one server is best in the extreme cases when traffic is very light, regardless of job size variability. So, not only is variability important, but so too is traffic intensity. This importance of traffic intensity is also noted by Scheller-Wolf [26], where he shows that under so-called power-law service times, moments of response time are functions of both the number of servers and the traffic intensity. Very recently, Molinero-Fernandez et. al. [20] consider the question of how many servers are best in an M/GI/k single priority system with a *heavy-tailed* service distribution. They approximate the service times with a bimodal distribution, obtaining an often good closed form approximation for the M/GI/k, under heavy-tailed G.

Since even for a single priority class, the question of the optimal number of servers does not appear to have been fully resolved, in this paper we will first characterize the optimal number of servers for the case of a single priority class (Section 3) and then determine the optimal number of servers for the case of two priority classes (remainder of paper), in both cases under *general* PH job size distributions.

3 Single priority class: How many servers are best?

In this section, we consider the simplified problem of just *one* priority class, determining the number of servers that minimize the mean response time. Although no exact analysis exists for the M/GI/k/FCFS queue, the M/PH/k/FCFS queue is easily analyzable via matrix analytic methods [15], as its Markov chain has a state space infinite in only one dimension. Figure 2 shows a picture of the Markov chain that we use for analyzing the M/PH/2/FCFS queue.

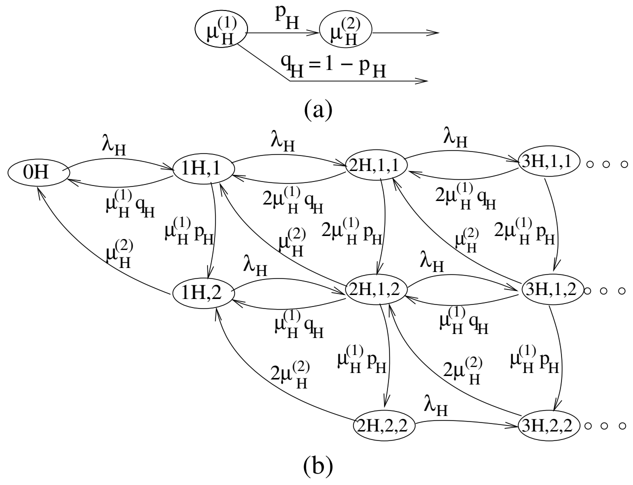
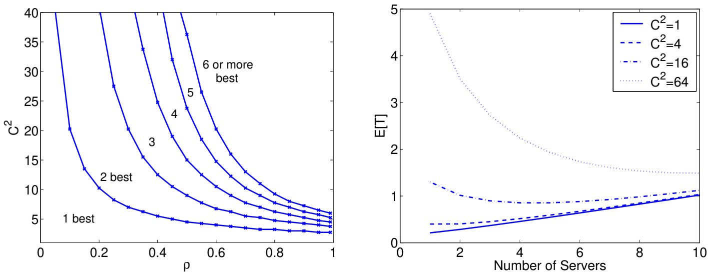


Figure 2: Figure illustrating that the M/PH/k/FCFS queue (single class – i.e., all jobs are high priority) is easy to analyze. (a) shows a 2-phase PH distribution with Coxian representation. (b) shows the Markov chain that is used to compute the mean response time for an M/GI/2 queue, where G has the representation shown in (a). The state represents the number of jobs in the system and the current phase of each job in service. The Markov chain is tractable because it is only infinite in one dimension.

Our results on the optimal number of servers are expressed as a function of the variability of the job size distribution, and the server load. While other factors, e.g., the exact form of the distribution might affect our results, we posit that load and variability will be the most relevant factors.

Figure 3(a) shows the optimal number of servers as a function of the load and the variability of the job size. Observe that under high job size variability and/or high load, the optimal number of servers is more than 1; we prefer k slow servers to 1 fast server. For example, at load $\rho = 0.4$ and $C^2 = E[X^2]/E[X]^2 - 1 = 20$, we see that 3 servers are best. Observe that computations are only done for up to 6 servers — the level curves shown will continue into the upper right portion of the plot if larger systems are considered.

Figure 3(b) shows that for any particular job size variability, $C^2 > 1$, having a larger number of slower servers may reduce the mean response time up to a point, after which further increasing the number of servers increases the mean response time. To understand why, note that by increasing the number of servers (while maintaining fixed total capacity), we are allowing short jobs to avoid queueing behind long jobs — specifically, an arriving short job is more likely to find a server free. Thus increasing the number of servers mitigates variability, hence improving performance. If the number of servers is too great however, servers are more likely to be idle, under-utilizing the system resources.



(a) (b)

Figure 3: The case of a single priority class. (a) The optimal number of servers as a function of the load, ρ, and the variability of the job size distribution, C^2. (b) Mean response time, $E[T]$, as a function of the number of servers at various job size variabilities ($C^2 = 1, 4, 16, 64$).

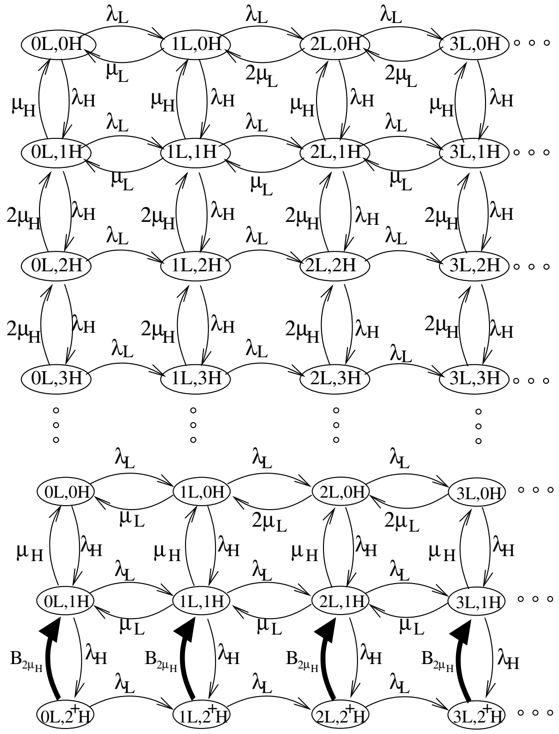


Figure 4: Markov chain for system with 2 servers and 2 priority classes where all jobs have exponential sizes. The top chain is infinite in two dimensions. Via the Dimensionality Reduction technique, we derive the bottom chain, which uses busy period transitions, and is therefore only infinite in one dimension.

4 The M/GI/k with dual priorities

In this section, we describe our analysis of the mean response time in M/PH/k/FCFS queues having two priority classes (high priority and low priority), where high priority jobs have preemptive-resume priority over low priority jobs. Since the mean response time of the high priority jobs can be analyzed as an M/PH/k/FCFS queue with a single priority as in the previous section, we concentrate here on the mean response time of the low priority jobs.

As mentioned in Section 2 the Markov chain for our system grows infinitely in two dimensions, which makes analysis via standard techniques intractable. In the case where the high priority job sizes are exponentially-distributed, the 2D-infinite Markov chain can be reduced to a 1D-infinite chain by applying the Dimensionality Reduction technique in [9]. Unfortunately, we will see that when high priority jobs have general PH job sizes, the technique in [9] alone does not suffice. We therefore need to augment the Dimensionality Reduction technique with the method of Neuts [22] for the calculation of lengths of various kinds of busy periods in M/PH/k queues. The combination of these two techniques enables us to, for the first time, analyze the M/PH/k queue with dual priority classes.

4.1 Exponential job sizes

We first consider the simplest case of two servers and two priority classes, where both high and low priority jobs have exponentially-distributed sizes with rate μ_H and μ_L respectively. Figure 4 (top) illustrates a Markov chain of this system, whose states track the number of high priority and low priority jobs, and hence grows infinitely in two dimensions. Observe that high priority jobs simply see an M/M/2 queue, and thus their mean response times can be computed using the analysis described in Section 3. However, low priority jobs sometimes have access to an M/M/2, M/M/1, or no server at all depending on the number of high priority jobs. Thus their mean response time is more complicated.

Figure 4 (bottom) illustrates the reduction of the 2D-infinite Markov chain to a 1D-infinite Markov chain via the Dimensionality Reduction technique. 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, the system switches into a mode where only high priority jobs are serviced until the number of high priority jobs drops to one. The length of time spent in this mode is exactly a single-server 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 $B_{2\mu_H}$.[3] The limiting probabilities of the bottom Markov chain in Figure 4 can be analyzed using the matrix analytic method. Given the limiting probabilities, it is easy to obtain the mean number of low priority jobs, which in turn yields their mean response time from Little's law.

Figure 5 shows the generalization to a 3-server system. We have added one row to the Markov chain shown in Figure 4, and we 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.

Our method is far simpler and more computationally efficient

[3] Matching three moments of busy period distributions is often sufficient to guarantee accurate modeling of many queueing systems with respect to mean performance [25].

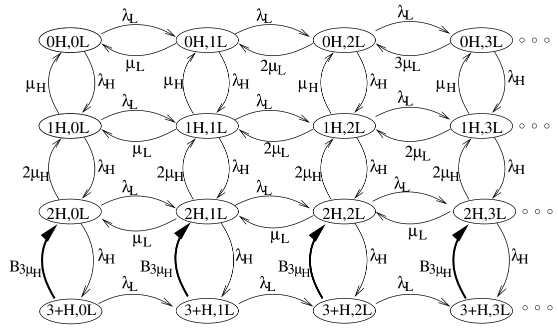


Figure 5: This Markov chain illustrates the case of 2 priority classes and 3 servers.

than methods shown in the prior published literature for analyzing the mean response time in an M/M/k queue with dual priorities (exponential job sizes). Furthermore our method allows general PH job sizes and is highly accurate; see validation Section 5. By contrast, the prior literature on the M/M/k priority queue (exponential job sizes) has sometimes resulted in numerically unstable computations [7, 8] or complicated partitions of the state space [18, 23].

4.2 Phase type job sizes

At first it appears that the Markov chain shown in Figures 4 and 5 should easily generalize to the case of PH service times. However for PH high priority jobs, we can no longer view the busy period, which starts when the number of high priority jobs goes from $k-1$ to k and ends when the it drops back to $k-1$, as simply an M/M/1 busy period with service rate $k\mu_H$. For example, in the case where the number of servers is two ($k = 2$) and the high priority jobs are represented by 2-phase PH distribution (as in Figure 2(a)), there are four different types of busy periods, depending on the phases of the two jobs starting the busy period and the phase of the job left at the end of the busy period. These busy periods are either (i) started with two jobs both in phase 1 and ended with a job in phase 1, (ii) started with two jobs both in phase 1 and ended with a job in phase 2, (iii) started with one job in phase 1 and another in phase 2 and ended with a job in phase 1, or (iv) started with one job in phase 1 and another in phase 2 and ended with a job in phase 2. (Note that a busy period can never start with two jobs both in phase 2.)

To analyze the mean response time of low priority jobs, we augment the dimensionality reduction technique with Neuts' algorithm [22] for analyzing busy periods. We first show how a 1D infinite Markov chain can be established via Dimensionality Reduction, assuming that we know the durations and probabilities of these different types of busy periods, and then show how these busy period quantities can be established via Neuts' work.

As is the case of exponential job sizes, the 1D-infinite Markov chain for PH job sizes tracks the exact number of low priority jobs but only differentiates between 0, 1, and 2-or-more high priority jobs for the case of two servers ($k = 2$). Figure 6 shows the level of the Markov chain, where the number of low priority jobs is i. In state (iL,0H), no high priority jobs are in system; in state (iL,1H,u), one high priority job in phase u is in system for $u = 1, 2$; in state (iL,2$^+$H,u, v), at least two high priority jobs are in system (we are in a high priority busy period), and the two jobs that started the busy period were in phases u and v, respectively, for $u = 1, 2$ and $v = 1, 2$. The four types of busy periods are labeled as B_1, B_2, B_3, and B_4; the durations of these busy periods are approximated by PH distributions matched to the first three moments of the distribution. Finally, $p_{(u,v),w}$ denotes the probability that a busy period started by two jobs in phases u and v respectively ends with a single job in phase w for $u = 1, 2$, $v = 1, 2$, and $w = 1, 2$. Note that the ending phase of the busy period can be determined probabilistically at that moment when the second high priority job arrives which starts the busy period. The length distribution is then modeled conditionally on this ending phase.

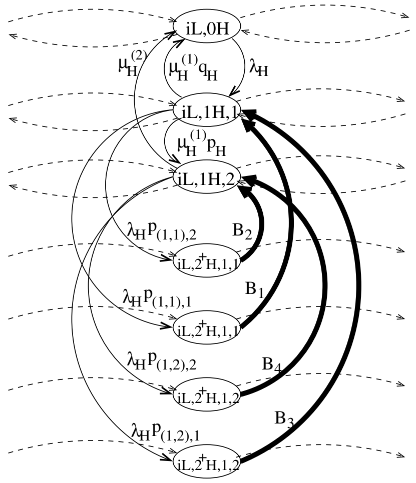


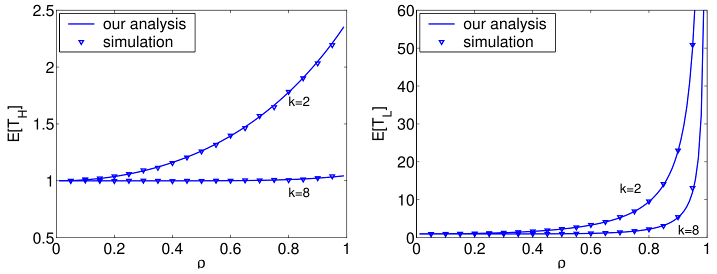
Figure 6: One level of the Markov chain for a system with 2 servers and 2 classes where high priority jobs have PH service times shown in Figure 2(a).

The remaining question is how to analyze the probabilities $p_{(u,v),w}$ and the first three moments of the duration of busy periods, B_1, B_2, B_3, and B_4. Observe that the Markov chain for high priority jobs (Figure 2(b)) is a QBD process, and the duration of different types of busy periods corresponds to the first passage time to state (1H,w) from state (2H,u,v) for different values of u, v, and w. Likewise, probability $p_{(u,v),w}$ corresponds to the probability that state (1H,w) is the first state reached in level 1 given that we start from state (2H,u,v) for $u = 1, 2$, $v = 1, 2$, and $w = 1, 2$. Neuts' algorithm establishes just these quantities, we defer the description to Appendix 8.

5 Validation and Comparison with Prior Work

The purpose of this section is twofold: (i) we validate our analysis against simulation, and (ii) we compare the results of our analysis to known approximation for the M/GI/k dual-priority queue. In addressing the above, we also begin our investigation of the factors

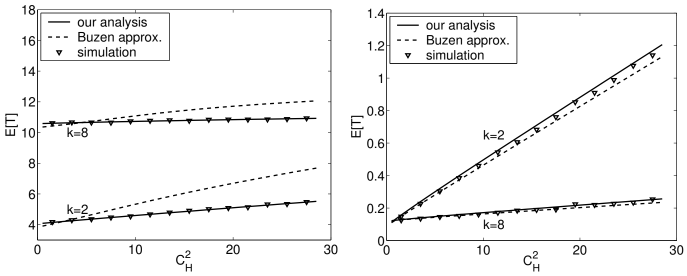
Validation



(a) High Priority (b) Low Priority

Figure 7: Validation of our analysis against simulation: two class M/PH/k system where $\rho_L = \rho_H$ and $E[X_L] = E[X_H] = 1$. The high priority jobs have a 2-phase PH job size distribution with $C^2 = 9$, shown in Figure 2(a), and the low priority jobs have exponential service times.

Validation Comparison: Mean response time



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

Figure 8: Comparison of our analysis with simulation and the Buzen-Bondi approximation for M/GI/k with dual priority. Only response times for low priority jobs are shown. (a) shows case when high priority jobs have a smaller mean job size than low priority jobs. (b) shows case when high priority jobs have a larger mean job size than low priority jobs. In these plots $\rho = 0.6$.

that affect the mean response time of high and low priority jobs.

In order to validate our algorithm, we compare our calculations with simulated results under a wide variety of loads and service distributions. Our simulations were calculated by running over 100,000 events over 30 iterations and averaging the results.

A subset of these validations are shown in Figure 7, for the case of 2 servers and 8 servers. As shown, our analysis matches simulation perfectly for both the high priority and low priority jobs. Because our analysis uses the matrix analytic method, the computational cost of the algorithm is well understood in terms of the size of the repeating matrices, which here is $k(5k+1)/2$ for a k server system. In practice this corresponds to a computation time of less than 0.1 seconds to calculate the mean response time of a system where $E[X_H] = E[X_L] = 1$ and $\rho_H = \rho_L = 0.25$ with $k = 4$.

It is interesting to compare the results of our analysis with the Buzen-Bondi approximation [1], which is the only published prior work to investigate the 2 class system under general job size distributions. Bondi and Buzen use the following very elegant and intuitive framework to approximate the performance of a priority system, where $E[W]^{M/GI/k/Prio}$ is the overall expected waiting time (response time minus service time) under the M/GI/k queue with priority classes.

$$E[W]^{M/G/k/Prio} = E[W]^{M/G/1/Prio}\left(\frac{E[W]^{M/G/k/FCFS}}{E[W]^{M/G/1/FCFS}}\right)$$

In our comparison, we use the exact results for $E[W]^{M/G/k/FCFS}$ (see Section 3); thus, the mean response time for the high priority jobs will be exact. However, when looking at low priority jobs, we still see the significant improvement of our algorithm over the Buzen-Bondi approximation.

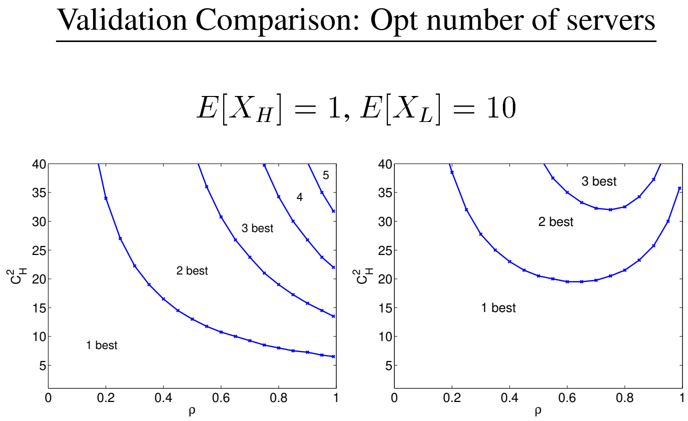
Figure 8 show the mean response time for low priority jobs predicted by our analysis, the Buzen-Bondi approximation, and simulation. Figure 8 (a) shows the case where high priority jobs have a smaller mean size than low priority jobs; and Figure 8 (b) shows the converse. Our analysis does significantly better than the Buzen-Bondi approximation, particularly when high priority jobs have a smaller mean size. As we increase the variability of the high priority job sizes, the error of the Buzen-Bondi approximation increases, while our analysis matches with simulation for all cases.

The consequence of the mispredictions made by the Buzen-Bondi approximation may be important. The error in predicting the response time can lead to a misleading conclusion on the optimal number of servers (a question we will investigate in detail in Section 6). Figure 9 shows the optimal number of servers given by our analysis as compared with that predicted by the Buzen-Bondi approximation. The Buzen-Bondi approximation is not only incorrect at higher loads and/or C^2 values, but also qualitatively misleading. As load increases, so should the optimal number of servers; an effect not true in the Buzen-Bondi approximation (Figure 9(b)).

In addition to validating our analysis, Figures 7 and 8 also illustrate some important factors affecting mean response time of high and low priority jobs: the system load, the variability of high priority customers, and the number of servers. Observe for example that for the job size distributions in Figure 8 (a) 2 servers are preferable, while in (b) 8 servers are preferable. The effects of these factors are interrelated and will be investigated in detail in the next section.

6 How many servers are best?

In the introduction to this paper we set out to answer three questions. We have already answered the first of these: "Are multiple cheap slow servers ever better than a single expensive fast server?" in the case of single priority class (Section 3) and in the case of dual-priority classes (Section 5). The superiority of multiple cheap servers only becomes apparent when job size variability is high. Thus having an analytical method which allows for general PH job



(a) Our analysis (b) Buzen-Bondi

Figure 9: Comparison of our results with that of Buzen-Bondi with respect to the question of "how many servers are best?" Case shown assumes the high priority jobs have a smaller mean job size. The load made up by high priority jobs equals that comprised by low priority jobs. Column (a) shows the results from our analysis. Column (b) shows the results from the Buzen-Bondi approximation.

sizes was required in order for us to address this question under dual-priority classes. The reason why multiple slow servers are preferable under high variability job sizes, is that they offer short jobs a chance to avoid queueing behind long jobs, which in turn lowers mean response time.

Having established that multiple servers is often desirable, there are two questions remaining:

- Does the answer to "how many servers are optimal" differ for the different priority classes? Also, does the overall optimal number of servers in a dual-priority system differ from the case where the classes are indistinguishable — that is when all jobs have been aggregated into a single priority class?
- How does the number of servers effect the overall mean response time, as well as that of low and high priority jobs?

We find that the answers to these questions depend on the relative sizes and relative proportions (loads) of the classes, as well as the variability of high priority jobs. We briefly summarize some of our findings. The number of servers preferred by low priority versus high priority jobs can vary widely. We perform a sensitivity analysis characterizing exactly where they disagree. Moreover, the number of servers preferred in the dual priority case when averaged over both classes typically differs substantially from the number preferred for the single class aggregate case. (This difference is not observed when the mean job sizes of high and low priority jobs are the same, but is observed elsewhere.) Furthermore, the absolute effect on mean response time can be dramatic (ranging from a factor of 2 to 6) as the number of servers is varied. Furthermore, for any highly variable job size distribution, there exists an "optimal" number of servers, where using fewer or more servers results in substantially worse performance.

6.1 Experimental setup for results graphs

We split up our evaluation into 3 cases, depending on the relative sizes of high and low priority jobs:

$E[X_H] = 1, E[X_L] = 1$ The mean size of high priority jobs equals that of low priority jobs.

$E[X_H] = 1, E[X_L] = 10$ The mean size of high priority jobs is smaller than that of low priority jobs.

$E[X_H] = 1, E[X_L] = 1/10$ The mean size of high priority jobs is larger than that of low priority jobs.

Throughout our evaluations, we will consider a range of variability in the high priority job sizes, typically shown on the y-axis, and a range of load typically shown on the x-axis. The variability of the low priority job sizes is held constant ($C^2 = 1$). Observe that variability in the low priority jobs is less interesting since the low priority jobs only affect each other under preemptive resume, and we've already studied the single class M/GI/k/FCFS queue. Lastly we also vary the proportion of the load made up by high priority and low priority jobs.

Some technicalities of the setup follow. In all the results shown, the high priority job sizes follow a 2-phase PH distribution with Coxian representation (Figure 2(a)), allowing any variability $C^2 \geq 1.5$. The mean job size for high priority jobs is held fixed at 1. When varying the proportion of load in each priority class, we vary the arrival rates of the classes only. In order to compare our results for the dual priority system to the same system having a single aggregate class, we combine the two-phase PH high priority job size distribution and the exponential low priority job size distribution to obtain the overall job size distribution aggregated across both classes.

6.2 Results

Figures 10, 11, and 12 illustrate our results for the three size cases above: $E[X_H] = 1$, $E[X_L] = 1$; $E[X_H] = 1$, $E[X_L] = 10$; and $E[X_H] = 1$, $E[X_L] = 1/10$ respectively. For each figure column (a) shows the case where the load made up by high and low priority jobs is equal, and column (b) shows the case where $\rho_H < \rho_L$. We also discuss, but omit showing, the case where $\rho_H > \rho_L$. For each figure we consider both the case of dual priority classes and the case of a single aggregate class.

Figure 10: Equal mean sizes

Looking at the topmost plot in Figure 10 column (a), we see that the high priority jobs do not always prefer one server. In fact in the case of higher variability and/or load, they may prefer five or more servers. This is to be expected based on our results in Section 3.

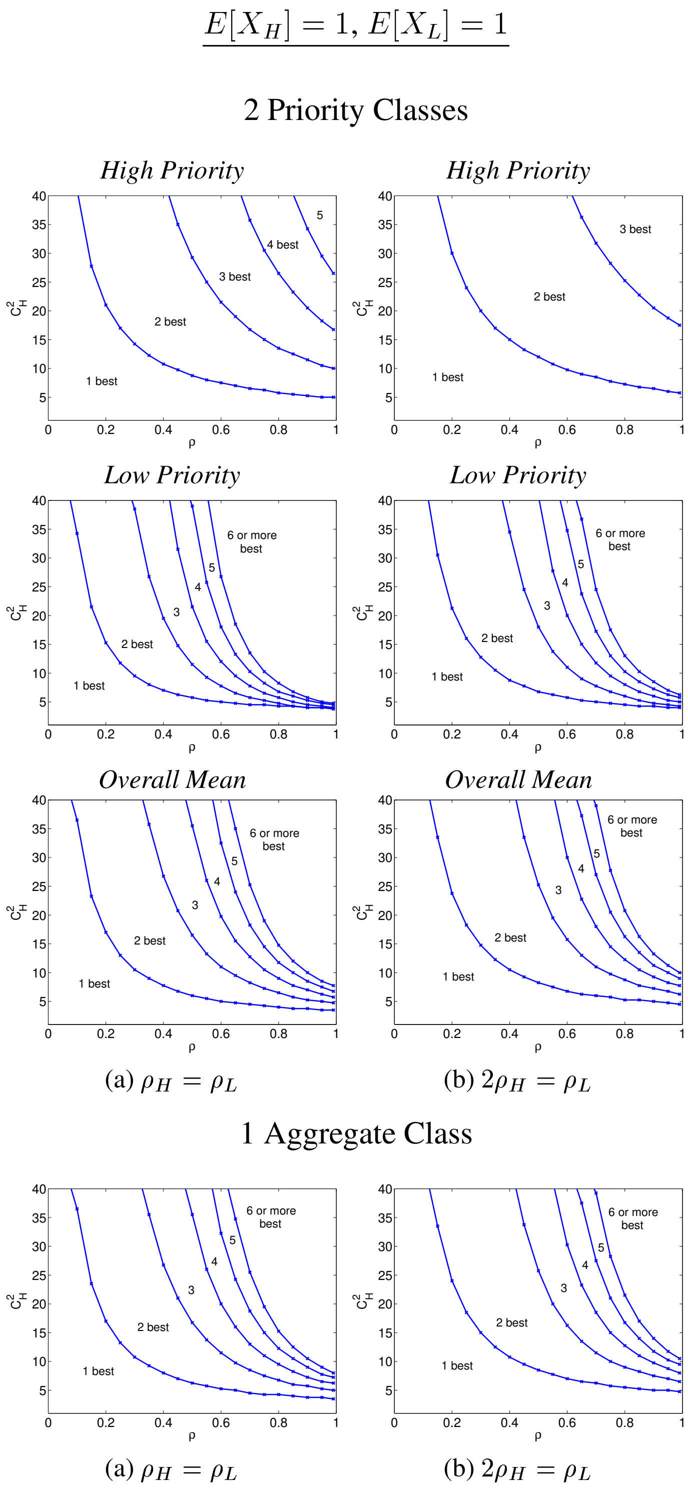


Figure 10: How many servers are best (with respect to mean response time) when the two priority classes have the same mean job size? Column (a) shows the case where the load made up by high priority jobs equals that of low priority jobs. Column (b) shows the case where the load made up by high priority jobs is half that of low priority jobs.

Surprisingly however, the number of servers preferred by low priority jobs (shown in the second plot in column (a)) is much greater than that preferred by high priority jobs. Although only up to six servers are considered in these plots, we will see in later plots (Figure 13(b)) that the difference in the number of servers preferred by low and high priority jobs can be more than 10 servers. Low priority jobs prefer more servers because low priority jobs are preempted by high priority jobs and thus their mean response time improves with more servers, which allows them to escape from the dominance of high priority jobs.

The preferred number of servers with respect to the overall mean response time (averaged over low and high priority jobs) is shown in the third plot in column (a), where we see that the number of servers preferred by the overall mean, as expected, is a hybrid of that preferred by low and high priority jobs. Note though that this hybrid is more weighted toward the preference of low priority jobs because adding extra servers only hurts high priority jobs a small amount; whereas adding extra servers helps low priority jobs enormously. Interestingly, the number of servers preferred with respect to the overall mean is nearly identical to that shown for a single aggregate class of high and low priority jobs, shown in the 4th (bottommost) plot in column (a). To understand why, observe that all jobs in this case have the same mean and thus prioritizing in favor of some of them over others does not affect the mean response time greatly. Even though the classes have different variabilities, that is a smaller-order effect.

Moving to column (b) of the same figure, we see that the same trends are evident when the high priority jobs make up a smaller fraction of the load. However, the specific numbers are quite different. For example, starting with the topmost plot in column (b), we see that the number of servers preferred by high priority jobs is much fewer. An explanation of this is that the high priority jobs only interfere with each other and they are fewer in number in column (b) than in column (a); thus they want fewer, faster servers.

Less obvious is the fact that the number of servers preferred by low priority jobs in column (b) is also fewer than that in column (a). This follows from the same reasoning; the low priority jobs are most strongly affected by preemptions from high priority jobs, and with fewer high priority jobs, there are fewer interruptions and thus fewer servers are needed to avoid queueing behind high priority jobs.

Since both the high and low priority jobs in column (b) prefer fewer servers than in column (a), it makes sense that their overall mean (shown in the third plot of column (b)) also indicates that fewer servers are desired. This third plot also matches the 4th plot in column (b) consisting of a single-class aggregation of high and low priority jobs, for the same reason explained above – that jobs have the same mean.

Not shown in Figure 10 is the case where high priority jobs comprise more of the load. In this case, both classes prefer more servers and, therefore, the mean of the two classes also prefers more servers. The reason for this is the converse of the above situation – there are more high priority jobs, therefore they see more interference and want more servers. Further, the low priority jobs are preempted more frequently by high priority jobs and therefore also want more servers to alleviate the effect. Again the single aggregate class looks very similar to the two priority class overall mean.

Figure 11: High priority class has smaller mean

Moving to Figure 11, we continue to hold the mean high priority job size at 1 and increase the low priority job size to 10. We consider this case to be the "smart" case in that the high priority jobs have a smaller mean size than the low priority jobs. Here, giving high priority jobs preference schedules the system more efficiently.

Notice that the preferred number of servers for the high priority jobs is identical to that in Figure 10 because the high priority job size distribution is unchanged. However, the number of servers preferred by low priority jobs is now very different: they almost always prefer only one server. This follows from the fact that there are very few low priority jobs; so there is unlikely to be more than one low priority job in the system at a time. Thus, low priority jobs prefer a single fast server.

The overall preferred number of servers, averaged over the two priority classes, is again a hybrid of the preferences of the two classes, but this time is biased towards the preferences of the high priority jobs because they are in the majority, implying a preference for fewer servers than the corresponding graph in Figure 10. Recall that adding servers is a way to help small jobs avoid queuing behind larger jobs. Since we are in the "smart" case, where small jobs have priority already, we do not need the effect of multiple servers. *Thus, "smart" priority classes can be viewed as a substitute for adding more servers.*

Comparing the overall preferred number of servers for the case of dual priorities with that preferred under a single aggregate class, we see that this time there is a significant difference in preferences. The single aggregate class prefers many more servers. This again is a consequence of the fact that "smart" prioritization is a substitute for increasing the number of servers.

Column (b) of Figure 11 illustrates the same graphs for the case where the high priority jobs comprise less of the total load. The trends are the same as in column (a); however the preferred number of servers is significantly smaller in all figures. This follows from the same argument as that given for column (b) of Figure 10. In the case (not shown) where high priority jobs make up a greater proportion of the total load, the number of servers preferred is, as before, always higher than in column (a).

Figure 12: High priority class has larger mean

In Figure 12 column (a), we once again hold the mean high priority job size fixed at 1 and now assume the low priority job sizes have a mean size of $1/10$. This case differs from the prior figure because now we are giving priority to the large job sizes: this is the opposite of "smart" prioritization. Consequently, many more servers will be needed in this case.

Once again, looking the topmost plot in column (a), we see that the preferred number of servers for high priority jobs is unaffected, since the high priority mean job size distribution has not changed. The low priority jobs, shown in the second plot of column (a), have

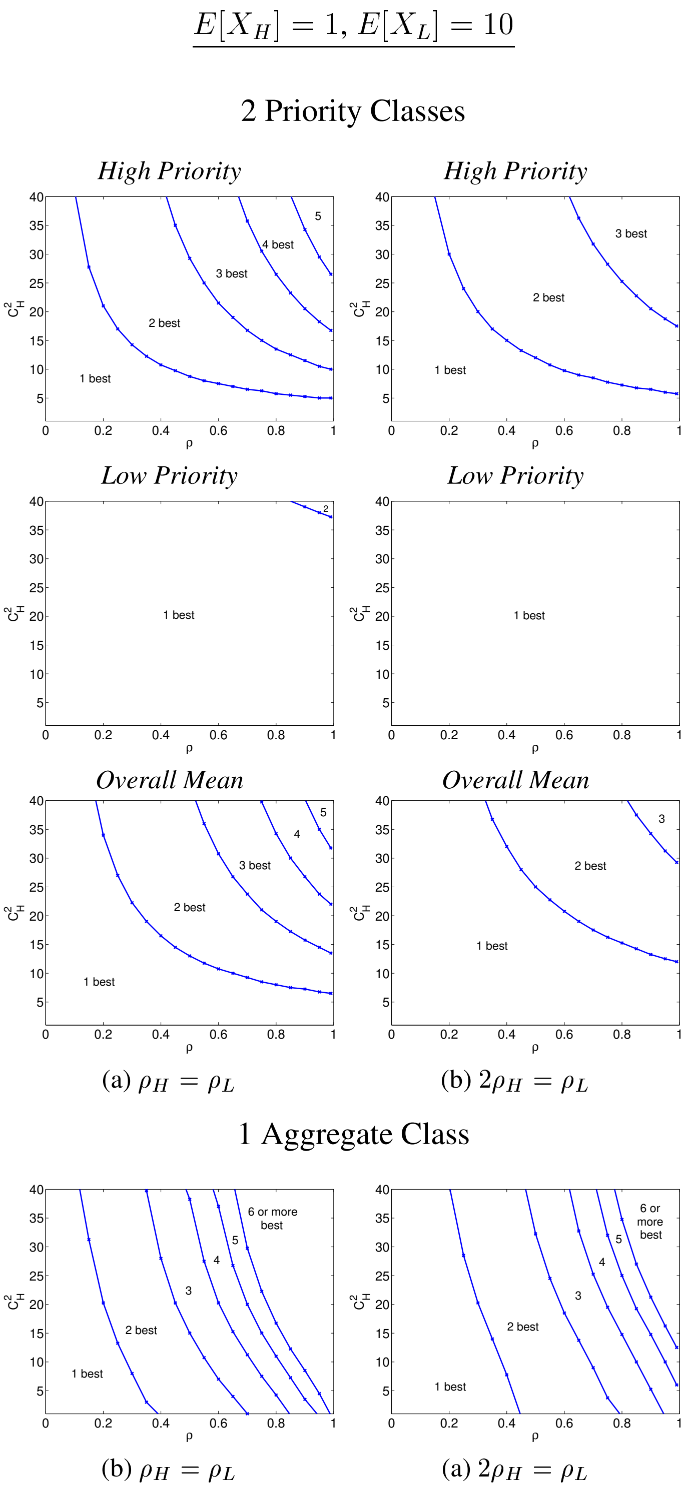


Figure 11: How many servers are best (with respect to mean response time) when the high priority jobs have a smaller mean job size? Column (a) shows the case where the load made up by high priority jobs equals that of low priority jobs. Column (b) shows the case where the load made up by high priority jobs is half that of low priority jobs.

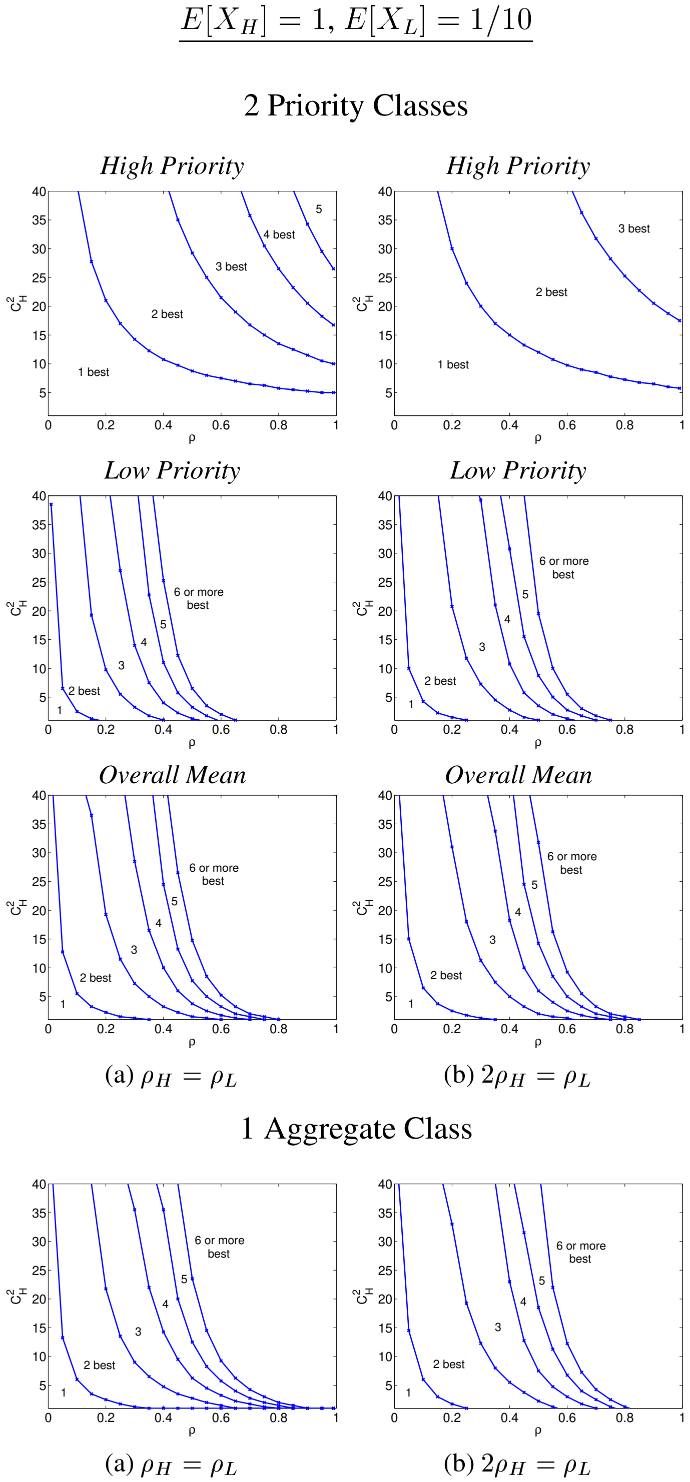


Figure 12: How many servers are best (with respect to mean response time) when the high priority class has a larger mean job size? Column (a) shows the case where the load made up by high priority jobs equals that of low priority jobs. Column (b) shows the case where the load made up by high priority jobs is half that of low priority jobs.

vastly different preferences from the prior case considered. Here the low priority jobs prefer a very large number of servers; whereas in Figure 11 they almost always preferred one server. Because the low priority jobs are very small compared to the high priority jobs, they want more servers in order to avoid being blocked, and forced to queue behind the large, high priority jobs.

The preferred number of servers for the overall mean response time in the dual-priority system, shown in the third plot of column (a), is again a hybrid of the preferences of the low and high priority jobs, but this time is strongly biased towards the low priority jobs because there are more of them. Notice therefore, that the number of servers preferred is much greater in this case. Comparing this with the single class aggregate, we see that the single class prefers slightly fewer servers than the dual class overall mean. This is due to the fact that the prioritization towards large jobs in the dual class system is not "smart."

Column (b) of Figure 12 illustrates the same graphs for the case where the high priority jobs comprise less of the total load. The trends are the same as in Column (a); however the preferred number of servers is significantly smaller in all figures. This follows from the same argument as that given for column (b) of Figure 10. In the case (not shown) where high priority jobs make up a greater proportion of the total load, more servers are preferable.

Figure 13: Time as a function of the number of servers

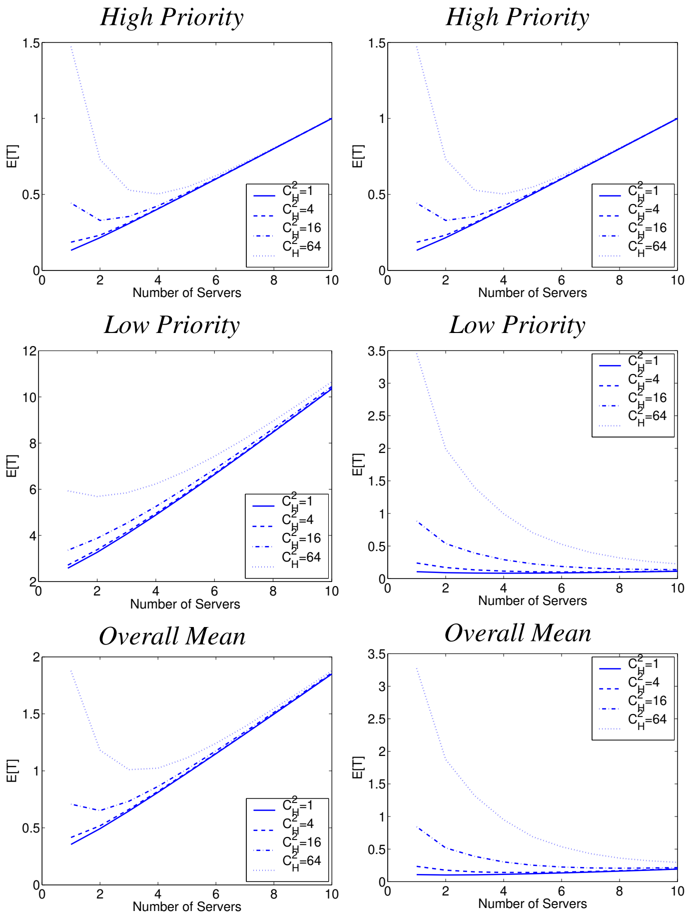
In all the prior results figures, we were concerned with determining the optimal number of servers as a function of system load and the variability of high priority jobs. Although we sometimes found k servers to be better than 1 server, we never looked at the actual mean response time as a function of the number of servers. In Figure 13 we do so, ranging the number of servers from 1 to 10. The key points made by this figure are that: (i) the mean response time of both priority classes is very sensitive to the number of servers and (ii) increasing the number of servers may reduce mean response time up to a point; however making the number of servers too large increases mean response time – thus forming a "U-shape." This figure also reinforces the prior message that *the greater the variability of the high priority jobs, the greater the number of servers needed to mitigate this variability*.

Structurally, Figure 13 is divided into two columns: column (a) considers the job size distribution shown in Figure 11 and column (b) considers the distribution shown in Figure 12. In the previous figures, we have already discussed at length the differences in the number of servers preferred by each class. This same information can be read off of Figure 13 by observing that each of the plots in the figure have a "U-shape" and the bottom of the "U" indicates the optimal number of servers.

Figure 13 however makes the following additional points. First, we see that, under high variability, the difference in the overall mean response time between the case of 1 server and the optimal number of servers is easily a factor of 2 in column (a) and, even more, close to a factor of 6 in column (b). Thus, variability does play a crucial role, imperfectly explored in prior research. Second, we see that, whereas in column (a) the optimal number of servers

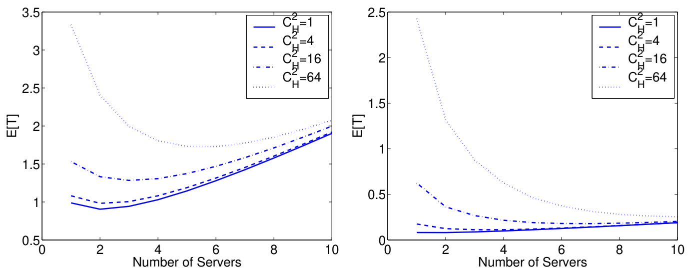
$\rho_H = \rho_L = 0.3$

2 Priority Classes



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

1 Aggregate Class



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

Figure 13: Mean response time as a function of the number of servers, which range from 1 to 10. 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 $\rho = 0.6$.

is quickly reached, in column (b) the optimal number of servers is in some cases greater than 10, not even appearing on the plot.

7 Conclusion

We have presented the first accurate analysis of the M/GI/k queue with dual priorities, which allows for arbitrary phase-type service time distribution. The accuracy of our results is always within 1%. Our method is conceptually simple: we transform a 2D-infinite Markov chain into a 1D-infinite Markov chain and then use Neuts' method to compute the various types of busy period durations needed in the 1D-infinite chain. Furthermore, our method is fast – requiring less than $.1 seconds$ per data point.

The speed of our method, combined with the fact that the method can handle general service times, allows us to address many questions in the area of multiserver queues that have not been addressable in prior work. In particular we are able to study the huge impact of server design (one fast server versus k slow servers) on mean response time, as well as the relationship between server design and job size variability. We find that when job size variability is high, many *cheap* slow servers are preferable to one *expensive* fast server, and the performance difference may be a factor of 2 – 6 in mean response time.

While our results apply to a single priority class as well, we feel that the results are most powerful and most interesting for the case of dual priority classes. Dual priority classes are a fact of life – some customers or jobs are simply more important, or have paid for better service agreements. Unfortunately, the design of multiserver systems in the case of dual priorities is quite complex. As we've seen, aggregating the dual classes into a single priority class does *not* allow us to predict the best system configuration (number of servers, given fixed total capacity) for minimizing mean response time. In fact, we've seen that the preferred configuration is dependent on the relative size of jobs in the high priority class versus that of the the low priority class, in addition to variability and load, and can vary wildly.

Another complexity in dual priority systems is that the preferred number of servers turns out to be quite different when the goal is minimizing mean response time of just the high priority jobs, or minimizing the mean response time of just the low priority jobs. As we've seen, depending on the workload, the number of servers preferred by low priority jobs may be an order of magnitude higher, or lower, than that preferred by high priority jobs. An interesting consequence of this fact with respect to designing systems, is that we can mitigate the penalty to low priority jobs (particularly the penalty caused by high variability of high priority job sizes), by choosing a server configuration which is more favorable to low priority jobs. We have seen that this can often substantially improve the mean performance of low priority jobs without being detrimental to high priority jobs.

A natural extension of the work in this paper is towards more than two priority classes. The simplicity of our analytical method makes it quite generalizable, and we believe that it can be applied to more priority classes as well. This is the subject of our current research.

References

[1] 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.

[2] 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.

[3] J. Calabrese. Optimal workload allocation in open queueing networks in multiserver queues. *Management Science*, 38:1792–1802, 1992.

[4] X. Chao and C. Scott. Several results on the design of queueing systems. *Operations Research*, 48:965–970, 2000.

[5] R. Davis. Waiting-time distribution of a multi-server, priority queueing system. *Operations Research*, 14:133–136, 1966.

[6] 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.

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

[8] 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.

[9] 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.

[10] Johnson and Taaffe. The denseness of phase distributions, 1988 – Manuscript.

[11] E. Kao and K. Narayanan. Computing steady-state probabilities of a nonpreemptive priority multiserver 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] G. Latouche and V. Ramaswami. *Introduction to Matrix Analytic Methods in Stochastic Modeling*. ASA-SIAM, Philadelphia, 1999.

[16] A. Mandelbaum and M. Reiman. On pooling in queueing networks. *Management Science*, 44:971–981, 1998.

[17] D. McWherter, B. Schroeder, N. Ailamaki, and M. Harchol-Balter. Priority mechanisms for OLTP and transactional web applications. In *International Conference on Data Engineering*, 2004 (to appear).

[18] 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.

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

[20] P. Molinero-Fernandez, K. Psounis, and B. Prabhakar. Systems with multiple servers under heavy-tailed workloads, 2003 – Manuscript.

[21] P. Morse. *Queues, Inventories, and Maintenance*. John Wiley and Sons, 1958.

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

[23] 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.

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

[25] 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.

[26] 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.

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

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

[29] S. Stidham. On the optimality of single-server queueing systems. *Operations Research*, 18:708–732, 1970.

8 Moments of busy periods in QBD processes

Neuts' algorithm [22] 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 [22] 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.

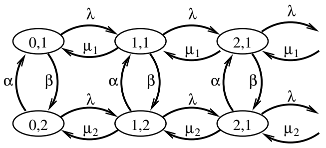
The purpose of this section is therefore to make Neuts' algorithm more accessible by re-describing his algorithm restricted to the first three moments of particular types of busy periods in QBD processes. We omit all proofs, which are provided in detail in [22], instead we focus on intuition and interpretation. 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 14 shows a particular QBD process with $m = 2$.



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

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.

8.1 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 s is $\leq x$ and the first transition out of state s is to state t. Observe 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 14. 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

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

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 [15]. The most straightforward (but slow) algorithm is to iterate

$$G = -A_1^{-1} A_0 G G - A_1^{-1} A_2 \tag{1}$$

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 $(1, j)$."

8.2 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 (1). Once G is obtained, matrix $G^{(1)}$ is obtained by iterating

$$G^{(1)} = \alpha_2^{(1)} + \alpha_1^{(1)} G + \alpha_1 G^{(1)} + \alpha_0^{(1)} G G + \alpha_0 G^{(1)} G + \alpha_0 G G^{(1)} \tag{2}$$

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

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

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

$$\begin{aligned} G^{(3)} &= \alpha_2^{(3)} \\ &\quad +\alpha_1^{(3)}G + 3\alpha_1^{(2)}G^{(1)} + 3\alpha_1^{(1)}G^{(2)} + \alpha_1 G^{(3)} \\ &\quad +\alpha_0^{(3)}GG + 3\alpha_0^{(2)}(G^{(1)}G + GG^{(1)}) \\ &\qquad +3\alpha_0^{(1)}(G^{(2)}G + 2G^{(1)}G^{(1)} + GG^{(2)}) \\ &\qquad +\alpha_0(G^{(3)}G + 3G^{(2)}G^{(1)} + 3G^{(1)}G^{(2)} + GG^{(3)}). \end{aligned} \tag{4}$$

We now give intuition behind expressions (2), (3), and (4). The right hand side of (2) 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 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 of the time to the first transition, and the second term, $\alpha_1 G^{(1)}$, is the contribution of the time it takes to reach $(0,k)$ after the first transition. Similarly, part [2] consists of three terms. The first term, $\alpha_0^{(1)}GG$, is the contribution of the time to the first transition, the second term, $\alpha_0 G^{(1)}G$, is the contribution of the time it takes to come back from level 2 to level 1 after the first transition, and the third term, $\alpha_0 GG^{(1)}$, is the contribution of the time it takes to go from level 2 to level 1.

The right hand sides of (3) and (4) 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 (3) and (4) can be interpreted exactly the same way as the three parts of (2) except that "the first moment" in (2) must be replaced by "the second moment" and "the third moment" in (3) and (4), respectively. The three terms in part [1] of (3) 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.

8.3 Generalization allowed

Finally, we mention some generalizations that Neuts' algorithm allows. (1) We restricted ourselves to the first three moments, but this can be generalized to any higher moments. (2) We restricted ourselves to the first passage time from level 1 to level 0, but this can be generalized to level i from level 0. (3) We restricted to QBD processes, but this can be generalized to M/G/1 type semi-Markov processes. (4) We restricted ourselves to the moments of the distribution of the duration of busy periods, but this can be generalized to the moments of the joint distribution of the duration of a busy period and the number of transitions during the busy period.