Inducing Optimal Scheduling with Selfish Users

Laurens Debo[1] • Paul Enders[1] • Anshul Gandhi[2]
Varun Gupta[2] • Mor Harchol-Balter[2] • Alan Scheller-Wolf[1]

[1] *Tepper School of Business*
Carnegie Mellon University, Pittsburgh, PA 15217, USA
{*laurdebo,penders,awolf*}*@andrew.cmu.edu*

[2] *School of Computer Science*
Carnegie Mellon University, Pittsburgh, PA 15217, USA
{*anshulg,varun+,harchol+*}*@cs.cmu.edu*

It is well known that scheduling jobs according to the Shortest-Remaining-Processing-Time (SRPT) policy is optimal for minimizing mean response time in a single-server system with online arrivals. Unfortunately, SRPT scheduling requires users to provide their job size (service requirement). Recent literature considers the problem of optimal scheduling when job sizes are either entirely unknown or partially unknown. We extend this work by additionally considering users who *know* their job size, but are *selfish* in that they are willing to lie about their size, if that affords them a delay advantage (e.g., pretending that their job is small).

It is not at all obvious how to schedule such selfish users, given that they are mixed in with users who genuinely don't know their size, and that the system administrator cannot differentiate between the two types. To address this question, we develop a novel approach for inducing users into self-scheduling themselves according to (an approximation of) SRPT and also revealing the size of their job.

We achieve this by defining a game that users play, whereby users are allowed to apply priority-boosting tokens to *portions* of their job. For a given number of tokens, we characterize the features of a unique equilibrium self-scheduling policy. We prove that the equilibrium policy is a dominant strategy, i.e. no user has any incentive to follow any other strategy, regardless of other users' behavior. Via an intricate queueing-theoretic analysis, we prove that this dominant strategy corresponds to a near approximation of SRPT scheduling when the appropriate number of tokens is distributed.

Distributing the appropriate number of tokens is known to be a difficult problem. Our queueing-theoretic analysis allows us to determine the optimal number of tokens to distribute to users, as well as allowing us to evaluate the effectiveness of our scheme as a function of job size variability, load, and other parameters.

Keywords: Strategic Queueing, Scheduling, SRPT, Equilibrium Analysis, Dominant Strategy, Selfish Users

1. Introduction

We consider online scheduling of jobs on a single machine. For this problem, it is well-known that running the job with the Shortest-Remaining-Processing-Time (SRPT scheduling), minimizes mean response time (Schrage 1968). A job's *response time* is the time from when it arrives until it completes. The optimality of SRPT holds regardless of the arrival times of the jobs and regardless of the sizes (service requirements) of the jobs – i.e., it holds for any arrival sequence. Thus any scheduler concerned with response time should strive to implement SRPT.

What is less well-understood is what to do when the job sizes aren't known. This may occur because a user genuinely *does not know* the service requirement of his job (the user is *uninformed*). Or, it may occur because an *informed* user (one who *does know* his service requirement) is *selfish:* willing to lie about his job size in an attempt to minimize his own response time. (For example, saying that his job size is small so as to gain advantage under SRPT scheduling). *We assume that both types of users – uninformed and selfish – may be present.* We are motivated by scheduling for supercomputing centers, where the center seeks to minimize mean response time, while users seek to minimize their own response times.

In operating systems, jobs of unknown size (e.g., Linux jobs) are typically handled by Round-Robin scheduling: jobs queue up to receive one small quantum of service from the processor, and then move to the back of the queue, where they wait again. In the limit, as the quantum size goes to zero (assuming no overhead) Round-Robin becomes Processor-Sharing (PS), a theoretical scheduling policy which says that whenever there are n jobs in the system they are each receiving exactly one-n^{th} of the processor. While PS has the advantage of not requiring knowledge of job size, it is nowhere near as effective as SRPT in minimizing mean response time. Furthermore, even when job sizes are unknown, it may be possible to improve upon PS. For example, if job sizes follow a distribution with decreasing failure rate, then the Foreground Background (FB) policy, in which the processor always works on that job that has received the least service so far, is optimal (Yashkov 1978, Yashkov 1987, Righter and Shantikumar 1989, and Feng and Misra 2003). However, FB is still outperformed by SRPT when job sizes are known.

Our problem is different from the above in that the scheduler does not know the job sizes (or their distribution), but some (selfish) users may know their job size, while others do not. This selfishness differentiates us from the growing body of work dealing with scheduling under inexact job sizes (e.g., Lu et al. 2004a, Lu et al. 2004b, Wierman and Nuyens 2008). Specifically, we assume: (i) There is 1 job per user; (ii) α fraction of users are uninformed and $1-\alpha$ fraction of users are informed (α may be unknown when operating the system); and (iii) all users are selfish. Within this context, we seek to schedule jobs as closely as possible to the optimal SRPT sequence.

One idea for achieving SRPT (and hence optimal) scheduling with selfish users is the policy we call "Shoot the Liar." The scheduler asks each user his job size, and then implements SRPT based on the declared sizes. If a job's size exceeds its declared size, the job is given lowest possible priority (it only runs when there are *no* other jobs in the system). The "Shoot the Liar" policy is certainly plausible, and in fact is implemented in many supercomputing centers. However it typically leads to an overestimation of job size by users, and thus sub-optimal scheduling (e.g. Chiang et al. 2002, Mu'alem and Feitelson 2001, Snell et al. 2002). It is also unfair to users who genuinely don't know their job size. We seek a policy that doesn't penalize users for not knowing their job size, while simultaneously providing SRPT-like scheduling in the presence of selfish users.

Our solution is very different from those proposed in the past: The users – instead of the system administrator – schedule their own jobs, following the rules of a game that we devise. Within our game we will show that if each user behaves selfishly, then the resulting schedule is close (in character and performance) to SRPT for all users who know their job size. In fact the

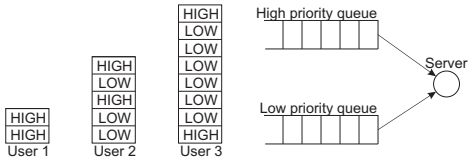


Figure 1: Three users playing our scheduling game. User 1's job consists of 2 quanta, user 2's job of 5 and user 3's of 8. Users are each given 2 tokens to place on 2 quanta of their choice, making these "high-priority" quanta. Jobs queue at the 2-queue system, shown on the right, where they receive only one quantum of service before returning to the back of the line until complete.

resulting schedule may be made arbitrarily close to SRPT as we increase the game's complexity. *Thus even with users behaving selfishly, we can induce SRPT scheduling, without knowing job sizes.*

We note that our solution does *not* involve the exchange of "real money," (having utility outside the system) and thus differs from Dolan (1978), Hassin and Haviv (1997), and Mendelson and Whang (1990) which use prices or Clarke tax based mechanisms. In many scheduling applications, e.g. scheduling at supercomputing centers, it is undesirable to use "real money" as processor-hours are already paid for by e.g. the NSF (see Chun et al. 2005). Prior work is detailed in Section 2.

Our basic game setting (described in detail in Section 3) is as follows: There are two queues, one high-priority, one low-priority. Jobs in the high-priority queue are always run, non-preemptively, before those in the low-priority queue. We assume all jobs are subdivided into equal-sized quanta which must be run sequentially, as shown in Figure 1. Upon arrival to the system, we give each user some number, say $D = 2$, of tokens. Then, *without any knowledge of the system state*, the user must place these 2 tokens on two of the quanta comprising his job. Quanta with tokens run in the high-priority queue, those without in the low.

Within this game, we assume the α fraction of uninformed users spend their tokens on their first D quanta: This is least wasteful of tokens, and also is in accordance with FB scheduling. We refer to this strategy as Immediate Gratification (IG), since these users are choosing to immediately spend their tokens. (We briefly discuss other policies for these users in Section 5.3.)

We then examine how the informed users should play the game so as to minimize their individual response times. Since the informed users are assumed selfish, we first characterize an equilibrium strategy (one that minimizes an individual informed user's response time assuming all other informed users are also minimizing their own response time). We then show that this equilibrium strategy is strongly dominant: irrespective of other users' strategies, the equilibrium strategy minimizes an individual user's expected delay for every possible game instance. It is crucial that we force users to place their tokens without observing the system state: We prove that if users are allowed to *observe* the state of the system prior to placing their tokens, a strongly dominant strategy cannot exist. Furthermore, we show that, in the unobservable case, the strongly dominant equilibrium strategy induces an approximation of SRPT scheduling, so-called *Two-level SRPT*,

Finally, we perform a detailed queueing-theoretic analysis of the equilibrium response time of our game. This enables us to answer several questions, such as how many tokens to allocate so as to minimize overall mean response time, and which factors most influence this allocation and overall system performance? We find that the proportion of informed customers, total system load (ρ, the ratio of mean jobsize to the mean interarrival time), and jobsize variability are the key determinants of system performance. We also see that our game can significantly reduce the gap between SRPT and Round-Robin (or PS) performance when there are sufficient informed users and the number of tokens is selected optimally.

2. Prior work

We bridge two large sub-areas within the field of job scheduling in order to minimize mean response time: (i) The case *without strategic users* in which jobs are scheduled using a global scheduling policy (e.g. SRPT, PS, RR or FB); and (ii) The case in which jobs are scheduled for *strategic users*.

No Strategic Users – Global Scheduling Policies. Prior work in single server scheduling has analyzed policies including SRPT, FB, RR and PS (and variants). The optimality of SRPT was proved as early as 1968 by Schrage. However, SRPT assumes that the exact service times of all jobs are known. More general situations are evaluated by Harchol-Balter et al. (2003) and Wierman and Nuyens (2008); they consider SRPT when one can only differentiate between n levels of job size (n-level SRPT). These papers differ in how jobs *within* each priority class are served: Harchol-Balter et al. (2003) assume FAIR (RR or PS) service, while Wierman and Nuyens (2008) consider the class of SMART policies, that excludes RR/PS. Nevertheless, both of these papers find that 2-level SRPT or 3-level SRPT already perform quite close to full-SRPT.

For unknown job sizes with a decreasing hazard rate FB is optimal; see Yashkov (1978) and Feng and Misra (2003). Righter and Shantikumar (1989) show that the tail of the distribution of number of jobs in queue is minimized under FB. For network flows, Aalto et al. (2004) and Avrachenkov et al. (2007) consider a two-level processor sharing (TLPS) model, with levels differentiated by attained service time. The latter paper presents analytic results for mean delay under TLPS; the former shows that PS is better than TLPS under an increasing hazard rate.

Strategic Users - Equilibrium Strategies. When considering strategic or selfish users, most research focuses on *delay costs* as a source of private user information, and users typically needing service for one, indivisible job. The objective of the system administrator is to maximize the net value the service facility generates. Different approaches have been used to study the system performance with strategic users. Some researchers perform numerical experiments assuming that users follow a particular algorithm (e.g. Gagliano et al. 1995), while others (e.g. Chun et al. 2005) study how real human subjects behave when put in such an environment. Finally, researchers (including ourselves), establish analytical insights of the system equilibrium.

Hassin and Haviv (1997) introduce a high and low priority queue; upon arrival, the user selects to which priority queue to route her job, after *observing* the system. Users pay a *price* to join the high priority queue. Hassin and Haviv (1997) show that threshold policies of the type "Buy priority when, upon arrival, you see more than j jobs in the system" are equilibrium strategies. Adiri and Yechiali (1974) study the same problem but with multiple priorities in an $M/M/1$ setting. Dolan (1978) considers the question of eliciting user's private delay costs. He shows that a user can maximize his utility by revealing his true delay cost if the price of high priority equals the marginal delay cost imposed on other users. This requires calculating these marginal delay costs and the imposition of different prices on different users.

Mendelson and Whang (1990) consider the case in which strategic users have private information about their delay costs but do *not* observe the system state upon arrival. They design a price mechanism that elicits users' true delay costs; users that have high delay costs pay a higher price to obtain priority. This incentive compatible mechanism optimizes the net value of the system.

An excellent review of this stream of work can be found in Hassin and Haviv (2003).

In most of the research above, each user arrives with one, indivisible job to the service facility. Furthermore, prices (or "real money") are part of the coordination mechanism. In our context, a job is divisible into smaller pieces to be processed in sequence. We also avoid the exchange of real money; this is often not realistic in supercomputing centers (Chun et al. 2005). Instead we use *tokens* as a coordination tool for the system administrator; virtual currency (tokens) have been used in different settings by e.g. Chun et al. (2005) and Mutz et al. (2007).

3. The Game Set-up

In this section we formally introduce our game: The arrival streams, job sizes and policies of the system administrator. Then we describe the strategies and utilities of both informed and uninformed users. As we assume uninformed users follow IG, it remains to establish the informed users' strategy. To do so we define the conditions for strategies to be a symmetric equilibrium, and introduce the notion of a strongly dominant strategy. In Section 4, we characterize the strongly dominant strategy for informed users in our game.

The arrival streams and job size. We consider a generic arrival process of users who each need one job to be processed. The arrival process is not affected by users' actions or the system state. Each user's job is composed of n *quanta* that each take one unit of time on the server and need to be executed consecutively; $n \in \mathbb{N}$ is a discrete random variable. None of the users observe the system state upon arrival. There are two types of users – *informed* users who know exactly how many quanta comprise their job, and *uninformed* users who have no knowledge of their job size.

We denote by Ω the set of all finite sample paths of arrivals. An $\omega \in \Omega$ is a finite sequence of arrival times, n (number of quanta) and user types (informed or uninformed). We define the probability measure $\mathcal{M}$ on Ω induced by the stochastic generation of samples paths ω; thus expectation with respect to $\mathcal{M}$, $\mathbb{E}_{\mathcal{M}}$, is well-defined. We assume $\mathcal{M}$ is common knowledge. (For brevity, we avoid rigorous measure-theoretic formalities such as σ-fields and measurable functions.)

The system administrator. The system administrator maintains one high priority and one low priority queue. The server depletes the high priority queue before proceeding to the low priority queue, without preemption. The system administrator distributes D tokens to each user upon arrival; each token routes one quantum of a job to the high priority queue. Upon arrival, all users need to submit a plan of how (on which quanta) the tokens will be spent *without* observing the system. For the service of his j^{th} quantum, the user joins the back of the high or the low priority queue – depending on the token placement – only after the $(j-1)^{st}$ quantum has been serviced.

The uninformed user's strategy. We assume an uninformed user always puts D tokens on the first D quanta of his job. Recall that we referred to this strategy as "Immediate Gratification."

The informed user's strategy: General framework. Informed user i's strategy is a vector of length n, $\boldsymbol{P}^i(n) = \left\{P_1^i, P_2^i, ...P_n^i\right\}$, with $P_n^i \in \{L, H\}$ and $\sum_{k=1}^{n} \delta_{\{P_k^i=H\}} \leq D$, where $\delta_{\{P_k^i=H\}} = 1$ if $P_k^i = H$ and 0 otherwise. Element P_j^i denotes whether user i assigns high or low priority to his j^{th} quantum. We refer to $\boldsymbol{P}^i(n)$ as user i's assignment of priorities. Let $\boldsymbol{P}^i = \left\{\boldsymbol{P}^i(1), \boldsymbol{P}^i(2), ...\right\}$ represent user i's strategy (i.e. for any realization of his job size). A *strategy profile* is the set of the strategies of all users. In Definition 1, we characterize *equilibrium* strategy profiles.

The user's objective. In our model, the informed users minimize their expected total response time to complete their job; this expected response time of any one user depends on the strategies of all other users. Since ex-ante, all informed users are the same, we can restrict our attention to symmetric strategy profiles in which all users play the same strategy. Consider a "tagged" user U, and let all other users u select the strategy $\boldsymbol{P}^u$. Then, upon observing n, user U's best response is

$$BR(n, \boldsymbol{P}^u) \doteq \arg\min_{\boldsymbol{P}^U} \mathbb{E}_{\mathcal{M}} \left\{M^U\left(n, \boldsymbol{P}^U(n), \boldsymbol{P}^u, \omega\right)\right\}$$

where $M^U\left(n, \boldsymbol{P}^U(n), \boldsymbol{P}^u, \omega\right)$ is the makespan on sample path ω of user U's job when all other users follow strategy $\boldsymbol{P}^u$, U's job size is n, and she assigns priorities $\boldsymbol{P}^U(n)$.

Definition 1 (Symmetric Equilibrium) *We say that the strategy profile in which all users play $\boldsymbol{P}^*$ is a symmetric equilibrium if $\boldsymbol{P}^*(n) \in BR(n, \boldsymbol{P}^*)$ for all $n \geq 0$.*

In other words, when the best response of an informed user with job size n to $\boldsymbol{P}^*$ is $\boldsymbol{P}^*$, the strategies are equilibrium strategies; all users utilize the same strategy.

Definition 2 (Strongly Dominant Strategy) *If* $\boldsymbol{P}^*(n) \in \arg\min_{\boldsymbol{P}} \left\{M^U(n, \boldsymbol{P}, \boldsymbol{P}', \omega')\right\}$, *for any* $\boldsymbol{P}'$, ω' *and* $n \geq 1$, $\boldsymbol{P}^*$ *is a strongly dominant strategy.*

A strongly dominant strategy minimizes a user's delay, irrespective of the strategy of all other users, *on every sample path.* A strongly dominant strategy is an equilibrium for a game with *any* measure $\mathcal{M}$ on arrival sample paths.

4. Analysis of the Game

Now, we define a specific strategy for an informed user, and show that it is strongly dominant:

Definition 3 (Delayed Gratification) *Delayed Gratification (DG) is the strategy which chooses high priority for the last* D *quanta, i.e.* $\boldsymbol{P}(n) = (\underbrace{L, L, ..., L}_{(n-D)^+}, \underbrace{H, H, .., H}_{\min(n,D)})$ *for all* $1 \leq n$.

Theorem 1 *DG is a strongly dominant strategy.*

From Theorem 1, we immediately obtain the Corollary:

Corollary 1 *The strategy profile where every user follows DG is our game's unique equilibrium.*

As a strongly dominant strategy, *DG* remains an equilibrium strategy for any arrival measure $\mathcal{M}$. Hence, *DG* – the *opposite* of *IG* – is a surprisingly compelling strategy, even for our *selfish* users to play. We briefly illustrate the intuition behind Theorem 1 and then proceed to its proof.

Consider two users A and B who arrive at time $t^A = 0$ and $t^B = \epsilon$ $(0 < \epsilon < 1)$, respectively. Both users have a job of size two quanta and $D = 1$ token. The system is empty at $t = 0^-$ and there are no further arrivals. There are only two possible strategies for each user: place the token on the first quantum (IG) or on the second (DG). The makespan (cost) matrix for this example is:

		User B: DG	User B: IG
User A	DG	3,4-ϵ	3,4-ϵ
	IG	4,3-ϵ	3,4-ϵ

We see that DG is the dominant strategy for both users: The entries of the makespan matrix are identical (i.e. 3,4-ϵ) except when user A follows IG but user B follows DG. This is because by joining the H queue on arrival, user A allows user B, who joins the L queue first, to *overtake* him in the L queue. Then, when user B departs the L queue she joins the H queue, and is immune from being re-overtaken by user A. The benefits of delayed gratification are thus two fold: (i) minimize overtake by later-arriving users in the L queue, and (ii) possibly overtake the L quanta of earlier-arriving users who are not following DG.

Sketch of the proof of Theorem 1 (See Appendix A for a complete proof): We prove that on *any sample path of arrivals,* ω, the makespan of a tagged user is minimized by *DG*, irrespective of the other users' strategies. Consider a fixed sample path of arrivals following any strategy $\boldsymbol{P}^u$, and two possible strategies $\boldsymbol{P}_{HL}$ and $\boldsymbol{P}_{LH}$ for tagged user U. The strategies $\boldsymbol{P}_{HL}$

and $\boldsymbol{P}_{LH}$ are identical except that one high/low priority pair, HL in $\boldsymbol{P}_{HL}$ for quanta j and $j+1$, is flipped to LH to obtain $\boldsymbol{P}_{LH}$. We will prove that U's makespan under $\boldsymbol{P}_{LH}$ is at most the makespan under $\boldsymbol{P}_{HL}$, for all n: $M^U(n, \boldsymbol{P}_{HL}, \boldsymbol{P}^u, \omega) \leq M^U(n, \boldsymbol{P}_{LH}, \boldsymbol{P}^u, \omega)$. This will prove the theorem since we can keep flipping HL to LH until we obtain DG.

The following proposition compares the state of the system at a specific time $t = \tau$ under strategies $\boldsymbol{P}_{LH}$ and $\boldsymbol{P}_{HL}$. Its proof appears in Appendix A.

Proposition 1 *Let τ be the first time after $t = 0$ when the tagged user U reaches the front of the H queue when following strategy $\boldsymbol{P}_{LH}$. Under strategies $\boldsymbol{P}_{HL}$ and $\boldsymbol{P}_{LH}$, at time $t = \tau$:*

1. *The set of users in the H queues in the two systems are identical. Further, these users are waiting on service for the same quanta in the two systems.*
2. *The set of users in the L queues in the two systems are identical. Further, these users are waiting for service for the same quanta in the two systems.*
3. *Under $\boldsymbol{P}_{LH}$, U is at the head of the H queue.*
4. *Under $\boldsymbol{P}_{HL}$, U is in the L queue.*

Using Proposition 1, it is easy to complete the proof of Theorem 1: As the arrival sequence and the subsequent strategy followed by user U after quantum $(j+1)$ are identical for $\boldsymbol{P}_{LH}$ and $\boldsymbol{P}_{HL}$, it follows from Proposition 1 that under $\boldsymbol{P}_{LH}$, U leaves the system no later than under $\boldsymbol{P}_{HL}$. As this result is independent of all other user's strategies, DG is a dominant strategy.

In the next lemma, we show that a system with only informed users ($\alpha = 0$), all using DG, is equivalent to a version of Two-level SRPT.

Lemma 1 *If all of the users in the system are informed ($\alpha = 0$) and using DG, our system is identical to Two-level SRPT.*

Proof As defined by Harchol-Balter et al. (2003), Two-level SRPT with cutoff η has two priority queues. All jobs with more than η units of processing time remaining join the low priority queue, and all jobs with no more than η units of processing time remaining (either upon arrival or after some processing has been completed) join the high priority queue. The low priority queue receives service only when there are no high priority jobs. Finally, within each queue service is provided according to what the authors call the FAIR discipline, which divides processing capacity evenly.

If we set $D = \eta$, then under DG if a job of size n arrives, the first $\max(n - D, 0)$ quanta of the job do not have tokens, and hence join the low priority queue: all jobs with more than η units of processing time remaining join the low priority queue. Once a job has η or fewer quanta remaining, all remaining quanta have tokens, and join the high priority queue. Finally, within each priority class jobs are served according to round-robin, which is FAIR. □

Our final theorem shows that forcing users to place tokens at arrival and without seeing the system state is necessary for the existence of a strongly dominant strategy. In the still-restricted setting in which users place tokens immediately on arrival but *after* observing only the queue lengths, our mechanism may break down.

Theorem 2 *If the game is modified so that users still place tokens immediately on arrival but after observing the number of users in each queue, then no strongly dominant strategy exists.*

Proof Consider the following 2-player game: Users A and B arrive to the system at time $t^A = 0$ and $t^B = \epsilon$ $(0 < \epsilon < 1)$, respectively. User A has a job of size $n^A = 3$ quanta and user B has a job of size $n^B = 4$ quanta; $D = 2$ tokens. The server is busy at $t = 0$ with a job that is serving its last quantum and leaves at $t = 1$. There are no other users in the system and no further arrivals, and the arrival sequence is common knowledge.

Users A and B are now involved in a dynamic game where A places tokens at time $t^A = 0$. If A plays (L,H,H), B observes the queue lengths $(n_H = 0, n_L = 1)$. However, if A plays either (H,H,L) or (H,L,H), B observes the queue lengths $(n_H = 1, n_L = 0)$. Thus user B only has partial information about user A's strategy and must form a belief about A's strategy based on what he observes. However, note that for the arrival sequence in this example, user B's utility (makespan) is independent of the strategy adopted by either player as long as A uses his tokens. Now, we will argue that this is not the case for user A, and the dependence of user A's utility on B's strategy precludes a strongly dominant strategy. Consider the following two possible strategies of user B:

$\boldsymbol{P}_1^B$ = Follow (L,L,H,H) irrespective of the observed queue lengths,

$\boldsymbol{P}_2^B$ = Follow (H,H,L,L) if there are users in the L queue, else follow (H,L,H,L).

User A's best response to $\boldsymbol{P}_1^B$ is $\boldsymbol{P}_1^A$ = (L,H,H), and that to $\boldsymbol{P}_2^B$ is $\boldsymbol{P}_2^A$ = (H,L,H). Thus there is no strongly dominant strategy for user A. □

5. Performance (Queueing) Analysis

In Section 4, we established that the informed users use DG, irrespective of the strategy adopted by uninformed users. Further, we assume that the uninformed users adopt IG. This completely specifies user behavior in our game. While this establishes *what* the users will do, it does *not* tell us what performance these users will experience, how many tokens should be distributed so as to optimize performance, how effective our game is at approximating SRPT, or which factors affect the optimal token distribution and performance.

To answer these questions, we develop a queueing-based algorithm to calculate mean system delay experienced by a DG or IG user, as a function of α, system load ρ, the coefficient of variation of the job sizes $C^2 \doteq Variance(JobSize)/E^2[JobSize]$, and the token endowment, D. This algorithm is a significant extension to the simpler algorithm used by Wolff (1970) to analyze round-robin scheduling. Compared to Wolff (1970) we track significantly more state variables and distinguish between different types of jobs, which change type while passing through the system.

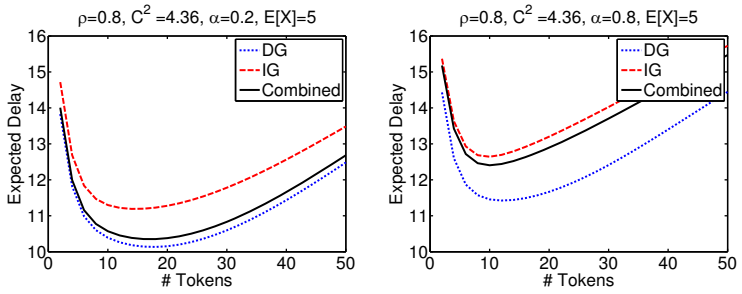
We briefly describe our algorithm in Section 5.1, before presenting graphs derived from application of our algorithm, in Section 5.2, to answer the questions posed above. We conclude this section with a brief discussion of the decision facing the uninformed users, and the relative merits of different strategies they might utilize, in Section 5.3

5.1 Algorithm for Calculating System Performance

We now briefly outline the queueing-theoretic algorithm we devise to calculate mean user delay; details are in Appendix B. We assume users arrive according to a Poisson process; each user brings an i.i.d. work requirement which may be broken up into a discrete, finite number of quanta, n.

We analyze mean delay, or time spent in queue, as a job's response time is equal to its delay plus its service time. We first decompose a user's total mean delay into the sum of the mean delay of each quanta. This decomposition is dependent upon: (i) whether the user is informed or uninformed (playing DG or IG); (ii) the user's total job size; and (iii) the token endowment.

Given this decomposition, we calculate the delay of each individual quanta through a series of interrelated recursive equations. These equations are derived by conditioning on (i) the system



(a) $\alpha = 0.2$. (b) $\alpha = 0.8$.

Figure 2: Expected delay vs Number of tokens.

state a "tagged user" sees upon arrival; and (ii) the characteristics of all users who arrive while the tagged user is in queue. These arrivals complicate the analysis significantly: If the tagged user is currently in the low priority queue and a new user arrives into the high priority queue, the tagged user must wait for service through the entire *busy period* initiated by this new user's high priority quanta, i.e. until the next time the high priority queue empties.

Simultaneously solving these recursive equations yields the mean delay for each quanta of the tagged user's job, conditional on the system state the tagged user sees upon arrival. We then "uncondition" (integrate over the possible system states the tagged user may see) utilizing the PASTA property, which yields the mean delay for the tagged user. We then uncondition again by integrating over the characteristics of the tagged user (IG or DG and her total job size) which finally yields the mean delay of a "typical" arrival to the system.

5.2 System Performance and Insights

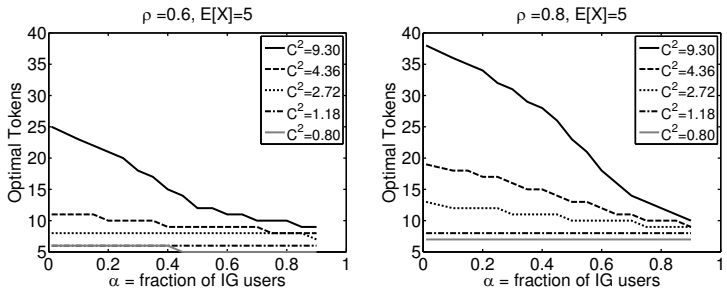
In this section, for tractability, we assume n is distributed as a mixture of geometric distributions.

Figures 2(a) and 2(b) show how the token endowment (horizontal axis) influences system delay of the DG and IG users individually, and in aggregate (vertical axis), for a representative problem instance ($\rho = 0.8, C^2 = 4.36$), for $\alpha = 0.2$, and $\alpha = 0.8$, respectively.

As expected, the delay of the DG users is strictly less than the delay of the IG users. Moreover, the mean delay of each user type (and in aggregate) decreases rapidly as the number of tokens is increased, before increasing more gradually as the token endowment surpasses optimal. This illustrates the fundamental dynamics of what we call the *token trade-off*: An increase in tokens shifts some quanta from the low priority to the high priority queue. This *increases* the delay of those users already in the high priority queue by interfering with their processing, but *decreases* the delay of the users who are shifted. Initially, as the endowment increases, the *gain* of the shifted users outweighs the *pain* inflicted on those users already in the high priority queue. But as the number of tokens grows, larger and larger users are included within the priority class, and congestion increases to the point that the *pain* outweighs the *gains* of increasing endowment.

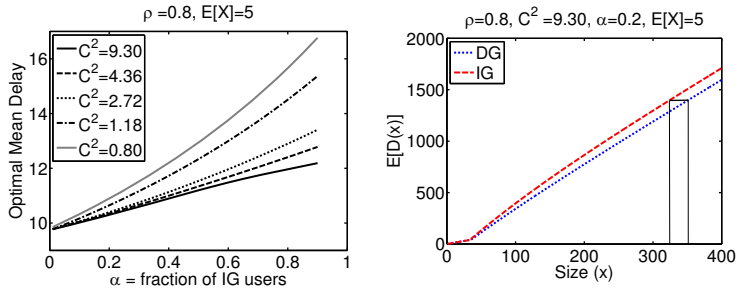
Figures 2(a) and 2(b) show that system improvement grows with the proportion of DG users: The DG users' jobsize knowledge allows them to utilize their tokens more efficiently. The *token trade-off* is better for DG users than for IG users, because only those arriving DG users with no more quanta than tokens immediately assume high priority, but *all* arriving IG users initially assume high priority. Thus the greater the proportion of DG users, the less relative *pain* for the same *gain*. Similarly, the DG users' delay is minimized at a slightly larger allocation than the IG users'; as their *token trade-off* is better the system would prefer to grant them larger allotments.

Note finally that with zero or a very large number of tokens prioritization vanishes (all quanta get the same priority) and the scheduling policy becomes round robin. At the optimal token



(a) $\rho = 0.6$. (b) $\rho = 0.8$.

Figure 3: Optimal number of tokens vs α.



(a) Optimal Delay vs α, $\rho = 0.8$. (b) Expected delay vs size.

Figure 4: Impact of α and Difference between DG and IG.

allocation, the system administrator can reduce delays by 15%-25% as compared to RR or PS scheduling for our examples: Tokens can be an effective tool at reducing system delays, *assuming the system administrator gets the allocation right.*

Figures 3(a) and 3(b) show how the optimal number of tokens, (vertical axis) decreases as the proportion of IG users grows (horizontal axis) for different values of job size C^2, for $\rho = 0.6$ and $\rho = 0.8$, respectively. These figures also show that as system traffic increases, or as the system variability grows, the optimal token allocation grows. Both of these effects spring from the the *token trade-off*: As load or variability grows, congestion in the low priority queue grows more rapidly than at the high priority queue, as only additional high priority users increase high priority queue congestion, but *all* additional users increase low priority congestion. Thus with increasing load or congestion, shifting more quanta to the high priority queue yields a net benefit; the relative *gain* increases faster than the relative *pain*, and optimal endowment consequently increases.

Figure 4(a) shows how the mean system delay given an *optimal* token allocation changes (vertical axis), versus the proportion of DG users (horizontal axis), for different values of C^2. Once again we see that systems with larger variability and larger proportions of DG users enjoy smaller comparable delays. Moreover, our policy reduces the effects of jobsize variability dramatically: As α approaches zero, the mean delay for all of the C^2 values converge to values very close to one another, between 9.77 and 9.85. As stated in Lemma 1, in this case our scheduling strategy becomes identical with *Two-level SRPT* as defined by Harchol-Balter et al. (2003). But how effective is this Two-level SRPT as a scheduling policy when $\alpha = 0$, especially compared to SRPT?

As expected, our policy is not as effective as SRPT, which yields mean delays between 5.73 ($C^2 = 9.30$) and 6.78 ($C^2 = 0.80$). But, our game claims between 71.70% and 76.76% of the possible improvements of SRPT over PS, which has a mean delay of 20 for all values C^2. And this is *using only two queues*, and *in the presence of selfish users.* By increasing the number of priority levels, queues, and accompanying currencies, our policy can be made arbitrarily close to SRPT.

5.3 Discussion of IG Users' Strategy

Thus far we have assumed that the uninformed users play the IG strategy; we briefly explore this assumption now, by asking what if a user knows that her job size lies within the range $[a, b]$? Could it be optimal to delay gratification and risk wasting tokens? We examine this question in Figure 4(b), which shows mean delay of both IG and DG against the user's jobsize for a representative example, under the optimal allocation for this example, $D = 35$.

Initially, for job sizes no larger than D the delays of IG and DG users are identical, as both user types spend all their tokens immediately. As job sizes grow DG begins to outperform IG. Crucial for our discussion is the *horizontal gap* between these two policies – see the line segment marked on Figure 4(b), linking the delay of an IG user of size 325 with the delay of a DG user of size 352.

If an *uninformed* user did in fact know that her job size was between 325 and 352, acting as though her job size were in fact 352 and using DG would lead to a *smaller* mean delay than using IG, as her delay would be upper-bounded by that of a DG job of size 352. (If her job size were less than 352 she would terminate earlier than she expects, wasting tokens but also reducing her delay.) This delay would be strictly smaller than the mean IG delay she would encounter. Thus it is beneficial for her to use DG, and potentially waste almost 80% of her tokens.

6. Conclusions

We consider the problem of online single-machine scheduling with a mixture of users: Users who *know* their job size and users who *genuinely don't*. We develop a game that motivates users with private information to truthfully reveal their job size, without unfairly punishing those users without such knowledge. Our game does so through user placement of a fixed number of tokens distributed to each user by the system administrator. We show that if users are unable to observe the state of the system, there exists a strongly dominant strategy in which the informed users place tokens to "boost" priority for the final portion of their job, Delayed Gratification (DG). By choosing to prioritize the final portion of their job (i.e. when remaining size is small), the users self-schedule themselves according to an approximation of SRPT. We show also that if users are able to observe the system prior to placing tokens, no strongly dominant strategy can exist.

Our mechanism differs in at least two ways from the mechanisms proposed in the literature. First, it elicits information from the informed users even in the presence of uninformed users. Second, we use a virtual currency that is of no value outside the system. Some existing papers also use virtual currency, but rarely consider the question of with how much currency to endow users.

We evaluate the performance of our mechanism via a queueing-theoretic analysis, calculating the expected delay and the optimal token endowment. We show that the optimal token endowment increases with the variability of job size and with system load, but decreases as the fraction of users who don't know their job size increases. These effects result from the relative trade-off incurred when increasing endowments – users who are granted priority experience a *gain* while those already with priority have their privilege diluted, experiencing *pain*. We also see that even with only 2 queues, our system approximates the optimal SRPT performance in the presence of mostly informed users, and converges to Two-level SRPT when all customers know their job size.

While in this paper we focus on users with either exact or no knowledge of their jobsizes, users typically have an estimate of their jobsize, which grows tighter as the job is processed. One way to utilize this knowledge is to allow users to place tokens in an as-you-go manner, which would also entail allowing them to observe the system while they are in queue. Developing and analyzing mechanisms to handle such users is significantly more complex, and planned future work.

References

Aalto, S., U. Ayesta, E. Nyberg-Oksanen. 2004. Two-level processor-sharing scheduling disciplines: mean delay analysis. 97–105.

Adiri, I., U. Yechiali. 1974. Optimal priority-purchasing and pricing decisions in nonmonopoly and monopoly queues. *Oper. Res.* **22** 1051–1066.

Avrachenkov, K., P. Brown, N. Osipova. 2007. Optimal choice of threshold in two level processor sharing. *ArXiv e-prints* **706**.

Chiang, S.H., A. Arpaci-Dusseau, M.K. Vernon. 2002. The impact of more accurate requested runtimes on production job scheduling performance. Feitelson et al. (2002).

Chun, B.N., P. Buonadonna, A. AuYoung, Chaki Ng, D.C. Parkes, J. Shneidman, A.C. Snoeren, A. Vahdat. 2005. Mirage: A microeconomic resource allocation system for sensornet testbeds. *Embedded Networked Sensors, 2005. EmNetS-II. The Second IEEE Workshop on* 19–28.

Dolan, R.J. 1978. Incentive mechanisms for priority queuing problems. *Bell Journal of Econ.* **9**(2) 421–436.

Feitelson, D.G., L. Rudolph, U. Schweigelsohn. 2002. *Job Scheduling Strategies for Parallel Processing*. Springer Verlag, Lect. Notes in Comput. Sci. vol. 2537.

Feng, H., V. Misra. 2003. Mixed scheduling disciplines for network flows. *ACM SIGMETRICS Perform. Eval. Rev. Special issue on MAMA 2003* **31**(2) 36–39.

Gagliano, R.A., M.D. Fraser, M.E. Schaefer. 1995. Auction allocation of computing resources. *Commun. ACM* **38**(6).

Harchol-Balter, M., B. Schroeder, N. Bansal, M. Agrawal. 2003. Size-based scheduling to improve web performance. *ACM Transactions on Computer Systems* **21** 207–233.

Hassin, R., M. Haviv. 1997. Equilibrium threshold strategies: The case of queues with priorities. *Oper. Res.* **45**(6) 966–973.

Hassin, R., M. Haviv. 2003. *To Queue or not to Queue*. Kluwer, Norwell, Ma, USA.

Lu, D., P. Dinda, Y. Qiao, H. Sheng, F. Bustamante. 2004a. Applications of srpt scheduling with inaccurate information. *Proceedings of the 12th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS* .

Lu, D., H. Sheng, P. Dinda. 2004b. Size-based scheduling policies with inaccurate scheduling information. *Proceedings of the 12th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS* 31–38.

Mendelson, H., S. Whang. 1990. Optimal incentive-compatible priority pricing for the m/m/1 queue. *Oper. Res.* **38**(5).

Mu'alem, A.W., D.G. Feitelson. 2001. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. *IEEE Trans. Prallel & Distributed Syst.* **12**(6) 529–543.

Mutz, A., R. Wolski, J. Brevik. 2007. Eliciting honest value information in a batch-queue environment. *Grid Computing, 2007 8th IEEE/ACM International Conference on* 291–297.

Righter, R., J.G. Shantikumar. 1989. Scheduling multiclass single server queueing systems to stocastically maximize the number of successful departures. *Probability in the Engineering and Informational Sciences* **3** 323–333.

Schrage, L.E. 1968. A proof of the optimality of the shortest remaining processing time discipline. *Oper. Res.* **16**(3) 687–690.

Snell, Q.O., M.J. Clement, D.B. Jackson. 2002. Preemption based backfill. Feitelson et al. (2002).

Wierman, A., M. Nuyens. 2008. Near-optimal scheduling despite inexact job-size information. *SIGMETRICS '08: Proceedings of the international conference on Measurement and modeling of computer systems*.

Wolff, R.W. 1970. Time sharing with priorities. *SIAM J. Appl. Math.* 566–574.

Yashkov, S.F. 1978. On feedback sharing a processor among jobs with minimal serviced length (in russian). *Technika Sredstv Svyazi. Ser. ASU* (2) 51–62.

Yashkov, S.F. 1987. Processor-sharing queues: Some progress in analysis. *Queueing Systems* **2** 1–17.

A. Equilibrium Analysis (from Section 4)

Proof of Theorem 1 Since in the static setting, the arrival instants and strategies followed by other users are decided independent of the system state, proving that DG is the dominant strategy is equivalent to proving that on every sample path of arrivals, the makespan of a tagged user is minimized by adopting DG.

Consider a fixed sample path of arrivals, a tagged arrival U, and two possible strategies $\mathbf{P}_{HL}$ and $\mathbf{P}_{LH}$ for U. The strategies $\mathbf{P}_{HL}$ and $\mathbf{P}_{LH}$ are identical except that one HL in $\mathbf{P}_{HL}$ is flipped to LH to obtain $\mathbf{P}_{LH}$. Let k be the index at which $\mathbf{P}_{HL}$ and $\mathbf{P}_{LH}$ first differ. We will prove that the makespan of U under strategy $\mathbf{P}_{LH}$ is at most the makespan under strategy $\mathbf{P}_{HL}$. This will prove the theorem since we can keep flipping HL to LH until we obtain the DG.

Let $t = 0^-$ be the time at which U finishes its $(k-1)$th phase and chooses to follow L under $\mathbf{P}_{LH}$ and H under $\mathbf{P}_{HL}$. Clearly, at $t = 0^-$ the state of the system in both cases is identical. The state description at $t = 0^-$ is given by:

n_H = number of users in H queue (excluding U)

n_L = number of users in L queue (excluding U)

$u_i^H \equiv$ user at the ith position in the H queue, $i \in \{1, \ldots, n_H\}$

$u_i^L \equiv$ user at the ith position in the L queue, $i \in \{1, \ldots, n_L\}$

$u^S \equiv$ user at the server

ϵ = excess of the user at server (this allows $k = 1$)

h_i^H = the length of the H streak (series of consecutive high priority jobs) in u_i^H's strategy (including and following the H phase it is waiting on)

h_i^L = the length of the H streak in u_i^L's strategy following the L phase its waiting on

h^S = the length of the H streak in u^S's strategy following the phase its serving at $t = 0^+$

Define $B^H(W, t)$ to be the length of the busy period initiated at time t by work of length W, which ends when the H queue becomes empty. That is, the busy period is the time to serve W, and the initial H streak of all the arrivals during the busy period.

Define $N_A^H(t_1, t_2)$ to be the number of external arrivals to the H queue in the time interval $[t_1, t_2)$. Define $N_A^L(t_1, t_2)$ to be the number of external arrivals to the L queue in the time interval $[t_1, t_2)$.

Let $T_H = B^H(\epsilon + h^S + \sum_{i=1}^{n_H} h_i^H)$

Let $T_B = B^H(\epsilon + h^S + \sum_{i=1}^{n_H} h_i^H + \sum_{j=1}^{n_L}(1 + h_j^L))$.

Let $m = N_A^H(T_B, T_B + 1)$.

Note that $t = T_H^+$ is the time at which under $\mathbf{P}_{LH}$, the first low priority job starts receiving service. Also, $t = T_B^+$ is the time at which under $\mathbf{P}_{LH}$, U starts receiving service for the L phase of interest. Further, $t = T_B + 1 + m$ is the time at which U starts receiving service for its following H phase under $\mathbf{P}_{LH}$ strategy. This is illustrated in Figure 5.

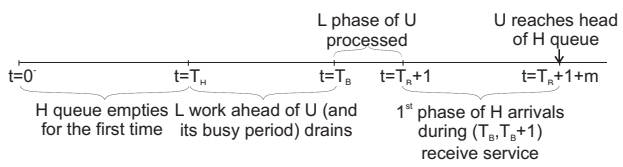


Figure 5: Illustration of the scheduling under $\mathbf{P}_{LH}$ strategy.

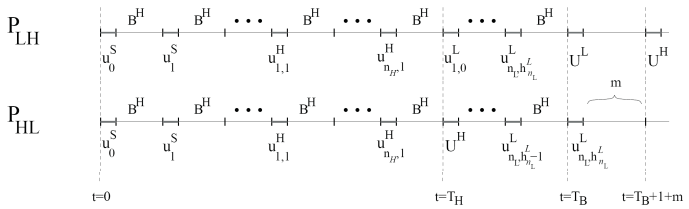


Figure 6: Illustration of the equivalence of state under $\mathbf{P}_{LH}$ and $\mathbf{P}_{HL}$ via reordering.

The following proposition compares the state of the system at $t = T_B + 1 + m$ under strategies $\mathbf{P}_{LH}$ and $\mathbf{P}_{HL}$.

Proposition 2 *Under strategies $\boldsymbol{P}_{HL}$ and $\boldsymbol{P}_{LH}$, at time $t = T_B + 1 + m$:*

1. *The set of users in the H queues in the two systems are identical. Further, these users are waiting on the same phases in the two systems.*
2. *The set of users in the L queues in the two systems are identical. Further, these users are waiting on the same phase in the two systems.*
3. *Under $\boldsymbol{P}_{LH}$, U is at the head of the H queue.*
4. *Under $\boldsymbol{P}_{HL}$, U is in the L queue.*

Using Proposition 2, it is easy to complete the proof of the Theorem. At $t = T_B + m + 1$, the state of the systems is identical except that under $\mathbf{P}_{LH}$, U is at the head of the H queue and under $\mathbf{P}_{HL}$, U is in the L queue (the order of the other users within the queues is immaterial). Further, the arrival sequence to the two systems as well as the subsequent strategy followed by U is also identical in the two systems. Therefore, under $\mathbf{P}_{LH}$, U leaves the system no later than under $\mathbf{P}_{HL}$.

Note that we did not impose the condition that users have the same number of jobs and tokens. Therefore the theorem is true in the case where users may vary in terms of demands and budgets. □

Proof of Proposition 2 The proof is illustrated in Figure 6. Define $B^H = B^H(1,t) - 1$ to be the busy period, of only high priority jobs, initiated at time t, by execution of a single job (of size 1).

Points 3 and 4 of Proposition 2 are relatively straightforward. To prove points 1 and 2, it will be helpful to consider a reordered scheduling of jobs in the two systems, as depicted in Figure 6. By $u^H_{i,j}$, we mean the jth H phase (relative to the state at $t = 0^-$) of user u^H_i. For the users present in the L queue at $t = 0^-$, $u^L_{i,0}$ denotes the L phase u^L_i is waiting on at $t = 0^-$, and $u^L_{i,j}$ for $j \geq 1$ denotes the jth, consecutive, H phase following the L phase.

We continue with the phase of the user already at the server at time $t = 0^-$ and then do not schedule the next job until the busy period of incoming H jobs is finished (this includes their leading H streak and they subsequently move to the L queue by the end of this busy period). Then we pick the next phase of this user or the first scheduled phase of the next user (if scheduled before $T_B + 1 + m$) and continue. The reordering preserves the final state (at $t = T_B + 1 + m$) of the jobs present in the system at $t = 0^-$, and of the incoming jobs. Further, it is clear that for any user, the final state (queue,phase) under the reorderings shown in Figure 6 are identical. In particular, at $t = T_B + 1 + m$,

1. All the n_H users and u^S present in the system at $t = 0^-$ finish their H streak and are in the L queue (if they have an L following).

2. All the arrivals into the H queue during $[0, T_B)$ finish their initial H streak and are in the L queue (if they have an L following).

3. All the arrivals into the H queue during $[T_B, T_B+1)$ finish their first H phase and are waiting in the queue corresponding to their second phase.

4. All the arrivals into the H queue during $[T_B+1, T_B+1+m)$ are in the H queue waiting for their first H phase.

5. All the n_L users in the L queue at $t = 0^-$ finish their L phase and the following H streak and are back in the L queue (if they have an L following).

6. All the arrivals into the L queue during $[0, T_B+1+m)$ are in the L waiting for their first L phase.

While this reordered scheduling changes the order of users in the queue in the final state, the aim of the construction is only to show the equivalence of the states of a user under $\mathbf{P}_{HL}$ and $\mathbf{P}_{LH}$ strategies followed by user U. The order can be easily determined, although for proving Theorem 1, points 3 and 4 in the statement of the proposition about the position of U suffice. □

B. Performance (Queueing) Analysis (from Section 5)

In Section 5 we outlined a queueing analysis of our proposed model. Here, we outline how to determine the expected delay (waiting time) of an informed user, using DG, $E[W_{DG}]$ and the expected delay of an uninformed user, using IG, $E[W_{IG}]$. For brevity, we will focus on $E[W_{DG}]$. The size of a job, i.e. the number of quanta, follows any discrete distribution with finite support, let M be the maximum number of quanta, and x the size of a job.

For a job of size $x > D$, we define the delay as the total time spent in queue. This is given by

$$W_{DG}(x) = W^L_{1,DG} + \ldots + W^L_{x-D,DG} + W^{LH}_{1,DG} + \ldots + W^{LH}_{D,DG}$$

where $W^L_{i,DG}$ is (the random variable for) the time spent in the low priority queue waiting for service of the i^{th} quantum and $W^{LH}_{i,DG}$ is (the random variable for) the time spent in the high priority queue waiting for the service of the i^{th} high priority quantum (that is, the $(x-D+i)^{th}$ quantum overall).

For a job of size $x \leq D$, we define the delay as the total time spent in queue. This is given by

$$W_{DG}(x) = W^H_{1,DG} + W^H_{2,DG} + \ldots + W^H_{x,DG}$$

where $W^H_{i,DG}$ is (the random variable for) the time spent in the high priority queue waiting for service of the i^{th} quantum.

Note that $W^L_{i,DG}$, $W^{LH}_{i,DG}$, $W^H_{i,DG}$ as defined above are independent of the size of the job. Thus obtaining expressions for the expectations of these quantities will allow us to compute the mean delay of a job conditioned on its size and hence the mean waiting for the system overall.

To obtain the above quantities we use standard queueing theoretic techniques. We first consider the arrival of an arbitrary job (which we will refer to as the tagged job) to the system and obtain the expression for the expected waiting time for each type of quantum conditioned on the state of the system (number of users of each {type, original size, remaining size}) at the arrival instant of this job (Subsection B.1). To complete the analysis, we relate the expected waiting time for each quantum to the expected number of users of each type in the system at the arrival instant of an arbitrary user via Little's Law and PASTA (Poisson Arrivals see Time Averages) (Subsection B.2).

B.1 Expected waiting time per quantum type

We begin with the simplest case, i.e. obtaining $E[W^{LH}_{i,DG}]$: Recall that $W^{LH}_{i,DG}$ is the (random variable for the) time spent in the high priority queue waiting for the service of the i^{th} high priority quantum (that is, the $(x - D + i)^{th}$ quantum overall). Note that this is for a job with $x > D$. Therefore, this job has gone through the low priority queue for some earlier quanta and migrated to the high priority queue. Let S be the random variable for the size of other users in the system. $W^{LH}_{i,DG}$ is made up of:

a. All DG jobs with $x \leq D$, that arrived after the tagged job started its last low priority quantum (at that point the high priority queue was empty) *and* that are still around when the tagged job starts waiting on its i^{th} quantum to be processed. More precisely: DG users of size $1 \leq S \leq D$ that arrive during $W^{LH}_{i-1,DG} + 1$, DG users of size $2 \leq S \leq D$ that arrive during $W^{LH}_{i-2,DG} + 1$, and so on up to all DG users of size $i \leq S \leq D$ that arrive during $W^{LH}_{0,DG} + 1$, where $W^{LH}_{0,DG} = 0$.

b. All IG jobs that arrived after the tagged job started its last low priority quantum *and* that are still around when the tagged job starts waiting on its i^{th} quantum to be processed. More precisely: IG users of size $1 \leq S$ that arrive during $W^{LH}_{i-1,DG} + 1$, IG users of size $2 \leq S$ that arrive during $W^{LH}_{i-2,DG}+1$, and so on up to all IG users of size $i \leq S$ that arrive during $W^{LH}_{0,DG}+1$.

We now consider a slightly more involved case, and see how to obtain $E[W^{H}_{i,DG}]$. Recall that $W^{H}_{i,DG}$ is (the random variable for) the time spent in the high priority queue waiting for service of the i^{th} quantum. $W^{H}_{i,DG}$ is made up of:

a. All jobs present in the high priority queue that, upon arrival of the tagged job into the system, had i or more quanta to go in the high priority queue. These include DG users of original size less than or equal to D ($S \leq D$), DG users of original size larger than D but remaining size less than or equal to D, and IG users.

b. All DG jobs with $S \leq D$, that arrived after the tagged job first entered the system *and* that are still around when the tagged job starts waiting on its i^{th} quantum to be processed. More precisely: DG users of size $1 \leq S \leq D$ that arrive during $W^{H}_{i-1,DG}+1$, DG users of size $2 \leq S \leq D$ that arrive during $W^{H}_{i-2,DG} + 1$, and so on up to all DG users of size $i - 1 \leq S \leq D$ that arrive during $W^{H}_{1,DG} + 1$.

c. All IG jobs that arrived after the tagged job first entered the system *and* that are still around when the tagged job starts waiting on its i^{th} quantum to be processed. More precisely: IG users of size $1 \leq S$ that arrive during $W^{H}_{i-1,DG} + 1$, IG users of size $2 \leq S$ that arrive during $W^{H}_{i-2,DG} + 1$, and so on up to all IG users of size $i - 1 \leq S$ that arrive during $W^{H}_{1,DG} + 1$.

d. If there was a job J in service when the tagged user arrived, the job J can contribute to $W^{H}_{i,DG}$ if it is one of the following types:

   i. A DG job processing its last low priority quantum, and thus migrating to the high priority queue to process the last D quanta.

   ii. A DG job with original size larger than D, and remaining size (number of quanta of the job at the server that have been processed yet) less than D but larger than $i-1$, thus processing a high priority quantum upon arrival of the tagged job.

   iii. A DG job with original size less than D, but remaining size larger than $i - 1$.

iv. An IG job with more than $i-1$ quanta in the high priority queue to go.

If $x > D$, the first $x - D$ quanta of a job go through the low priority queue. $E[W^L_{i,DG}]$ is (the random variable for) the time spent in the low priority queue waiting for service of the i^{th} quantum. The delay of this quantum can be determined in a similar fashion as $E[W^H_{i,DG}]$ but there are additional complications. For brevity we do not go into the details of the delay analysis here but merely outline the main factors:

- The completion time of all jobs in the high priority queue that the tagged job observes upon arrival *and* all job that arrive into the high priority queue before it first empties. This is referred to as a busy period.
- The aforementioned busy period consisted of DG and IG jobs. Some of the IG jobs (those of size larger than $D + 1$) have migrated to the low priority queue and are behind the tagged job. But, these may cause delays for future low priority quanta of the tagged job.
- While the tagged job is waiting, busy periods of high priority jobs are created by new arrivals into the high priority queue as well as DG jobs migrating to the high priority queue. DG jobs may also arrive into the low priority queue. The IG arrivals into the high priority queue may, in turn, spawn low priority quanta joining the low priority queue.
- The job at the server that may contribute different amounts of waiting time depending on its type and how far the job has progressed.

So far we have focused on DG jobs. For uninformed users, that use IG, the analysis is similar. First, note that there are no IG jobs migrating from low to high priority. Second, as DG ($x \leq D$) and IG arrivals into the high priority queue observe the same system and the same delay we know: $E[W^H_{i,DG}] = E[W^H_{i,IG}]$. Third, the delay analysis for the low priority quanta of an IG job is similar but more involved than the delay analysis for the low priority quanta of the DG jobs.

B.2 Relating waiting time to number in system

The delay of any quantum depends on the system state observed be the job when it first arrives to the system. As job arrivals follow a Poisson process we can invoke PASTA (Poisson Arrivals See Time Averages). Therefore the tagged job observes the average state of the system upon arrival.

Little's law relates the mean number in system to the expected time in system (waiting time) given the arrival rate. Thus the mean system state is related to the waiting time of each quantum. Therefore both the mean number of jobs (in various stages of progress through the system) and the expected waiting time enter into a system of simultaneous equations.