Controlling Queues with Constant Interarrival Times

Esa Hyytiä*, Guðmundur Magńusson* and Rhonda Righter†
Department of Computer Science*, University of Iceland
Department of Industrial Engineering and Operations Research†, University of California Berkeley

***Abstract*—We consider server systems with constant inter-arrival times subject to arbitrary cost functions. This type of systems arises when we have full control over arrivals. Typical examples include situations where computers or network elements schedule periodic updates at regular time intervals (cf. cron daemon in unix systems), a congestion avoidance or load balancing mechanism imposes regular inter-arrival times at a lower level, and also in customer service and healthcare systems where patients book appointments. In the basic case, known as the D/M/1 queue, there is a single server and the service times are independent and exponentially distributed. We study different value functions for the D/M/1 queue that characterize the expected cost difference in the infinite time horizon if the system is initially in a given state instead of being in equilibrium. When the arrival process is Poisson, the corresponding results are compact and known. The fixed interarrival times complicate the situation, and even the mean waiting time is harder to characterize. We apply our results to develop a heuristic for a dispatching problem, and evaluate the heuristic numerically.**

***Index Terms*—D/M/1 queue, MDP, deadline, response time**

I. Introduction

Servers preceded with buffers or queues arise in numerous applications. Sometimes they are invisible to users, e.g., in large cloud computing systems, and sometimes they are clearly visible to us, such as waiting places and physical queues in health clinics or taxi stands. Of course, in the former case the end user experiences long queues as *unexpected delays*, deteriorating the user experience of the service.

In this paper, we first consider a single-server queue with constant interarrival times, and then apply our results to systems with multiple servers. The constant inter-arrival times tend to appear when we have full control over arrivals. Interestingly, from the analysis point of view, a constant inter-arrival time tends to be somewhat more challenging than the usual assumption of having a Poisson arrival process.

Server systems receiving jobs or customers at (near) constant inter-arrival times appear in different settings:

1) Constant inter-arrival times can result directly from the system design, e.g., when periodic tasks need to be executed. Typical examples includes situations where computers or network elements schedule periodic updates at regular time intervals (cf. cron daemon in unix systems).
2) Similarly, an admission control mechanism in a multi-server system may generate (approximately) constant inter-arrival times to individual servers. For example, timers can be used to implement a form of congestion avoidance (gapping), which helps to stabilize the system under high load.
3) In large systems of parallel servers, load balancing may lead to (approximately) constant inter-arrival times. In particular, with the Round-Robin dispatching policy, the inter-arrival times to each server are asymptotically constant.
4) Booking systems (at different time scales), where appointments or resources can be reserved in advance based on time slots, are prime examples of constant inter-arrival times. For example, in the healthcare sector patients make reservations for doctor appointments, and the arrival pattern thus has constant inter-arrival times.

The basic single-server system with independent and exponentially distributed service times is known as the D/M/1 queue. We focus on the D/M/1 queue subject to arbitrary costs and study the *value functions* that characterize the expected cost difference over an infinite time horizon if the system is initially in a given state instead of being in equilibrium. When the arrival process is Poisson, the results corresponding to the mean response and waiting time are compact and known. Any other interarrival distribution tends to complicate the situation. For example, Erlang-distributed interarrival times result when the round-robin routing is applied to a Poisson arrival process. The Erl/G/1 queue was analyzed in [1]. Similarly, fixed interarrival times complicate the situation, and even the mean waiting time is harder to characterize. Note that the D/M/1 queue is obtained as the large system limit under round-robin routing when the number of servers tends to infinity.

The main contributions of this paper are two-fold. First, we provide means to develop cost-aware heuristic yet efficient policies to control the aforementioned systems of parallel servers. Second, like a "cartographer", we explore the theoretical terrain of value functions related to queueing systems. As mentioned, a large body of past work, as is customary in queueing theory, assumes a Poisson arrival process. The lack of memory of the exponential inter-arrival times then facilitates the analysis and (i) value functions often have compact short forms, and (ii) are easy to utilize when developing policy iteration based dispatching policies. In this paper, we take an "orthogonal" step and vary the arrival process (instead of the service time distribution). This complicates both the derivation of the value function (for a single queue), and its application to systems of parallel queues.

The rest of the paper is organized as follows. First, in Section II we introduce our model and notation, and give a recursive method to compute the value function for the D/M/1 queue subject to arbitrary cost structure. In Section III, we consider rescheduling and dispatching problems, and in

Section IV we give some numerical examples. Section V concludes the paper.

II. Number-aware D/M/1 queue

In this section, we consider the D/M/1 queue, where jobs arrive at constant inter-arrival times β and their service times are independent and exponentially distributed with parameter μ. Hence, the offered load is

$$\rho = \frac{1}{\mu\beta}.$$

The stability condition $\mu\beta > 1$ says that a busy server should process more than one job per time slot on average.

It turns out that the D/M/1 queue is analytically tractable. Due to the lack of memory property of the exponential distribution, it is easy to see that the number of potential departures, D, during time β obeys a Poisson distribution with parameter $\mu\beta$.

We consider the so-called *number-aware* system, where the state is defined by the number of jobs (in contrast to size-aware systems, where exact service times are known). In particular, we let X_t denote the state *upon the arrival of the t^{th} job* (excluding the new job). It is easy to see that X_t constitutes a Markov chain with state space $\{0, 1, 2, \ldots\}$,

$$X_{t+1} = (X_t + 1 - D_t)^+,$$

where $D_t \sim \text{Poisson}(\mu\beta)$. From state n, the system can move to any state in $\{0, \ldots, n+1\}$, depending on how many of the present $n+1$ jobs finish before the next arrival. Let $p_n = \text{P}\{D = n\}$ and $g_n = \text{P}\{D \geq n\}$. Then the transition probability matrix of the embedded chain X_t is

$$\mathbf{P} = \begin{pmatrix} g_1 & p_0 & 0 & 0 & 0 & \\ g_2 & p_1 & p_0 & 0 & 0 & \cdots \\ g_3 & p_2 & p_1 & p_0 & 0 & \\ g_4 & p_3 & p_2 & p_1 & p_0 & \\ & & \vdots & & & \ddots \end{pmatrix},$$

where

$$\begin{cases} p_n = \dfrac{(\mu\beta)^n}{n!}\, e^{-\mu\beta} \\ g_n = \displaystyle\sum_{j=n}^{\infty} p_j. \end{cases}$$

It turns out that the mean waiting time and steady state distribution[1] are available for the D/M/1 queue. The stationary distribution of the D/M/1 queue, for the number of jobs seen by an arrival, X_t, is

$$\pi_n = (1-\delta)\delta^n, \quad \text{where } n = 0, 1, 2, \ldots$$

where δ is the root of

$$x = e^{-\mu\beta(1-x)}, \tag{1}$$

with the smallest absolute value [2].[2] For comparison, the stationary distribution seen by arriving jobs in the M/M/1 queue has the same geometric form,

$$\pi_n = (1-\rho)\rho^n, \qquad \text{where } n = 0, 1, 2, \ldots,$$

which is also the steady state distribution a random observer sees, cf. Poisson Arrivals See Time Averages property (PASTA).

The mean waiting time in the D/M/1 queue is

$$\text{E}[W_{\text{D/M/1}}] = \frac{\delta}{1-\delta} \cdot \frac{1}{\mu}, \tag{2}$$

Hence, δ corresponds to "an effective load" as the mean waiting time in the M/M/1 queue (with Poisson arrival process) is

$$\text{E}[W_{\text{M/M/1}}] = \frac{\rho}{1-\rho} \cdot \frac{1}{\mu}. \tag{3}$$

These are illustrated in Figure 1, where the benefits of a deterministic arrival pattern become obvious. For example, $\text{E}[W_{\text{D/M/1}}] < (1/2)\,\text{E}[W_{\text{M/M/1}}]$ for all $\rho < 1$.

A. Cost structures

Cost structures can often be defined in different yet equivalent ways. We associate the so-called immediate cost to arrival instants.

Definition 1 (Cost structure)**.** *The system incurs cost $c(n)$ when an arriving job sees the queue in state n, i.e., there are n jobs ahead, and the new job will be the $(n+1)^{st}$ in the queue.*

In principle, the $c(n)$ can be an arbitrary function, but usually it is non-negative and non-decreasing in n.

Example 1 (Mean times)**.** *An appropriate cost functions for sojourn time (response time) and waiting time (in queue) are*

$$c_t(n) = \frac{n+1}{\mu}, \qquad \textit{(sojourn time)} \tag{4}$$

$$c_w(n) = \frac{n}{\mu}, \qquad \textit{(waiting time)} \tag{5}$$

and the corresponding mean costs we already know are

$$r_t = \frac{1}{(1-\delta)\mu},$$

$$r_w = \frac{\delta}{(1-\delta)\mu}.$$

Example 2 (Deadlines)**.** *Response times exceeding some thresholds tend to lead to unsatisfactory user experiences and even income losses [4]. Suppose that whenever the response time of a job exceeds time τ a deadline violation occurs and a fixed unit penalty cost is incurred [5], [6]. Given exponential*

[1] The global balance equation for an arbitrary state $n \geq 1$ can be written as $p_0\pi_n = \sum_{k=2}^{\infty} g_k \pi_{n+k-1}$. Substituting a geometric trial, $\pi_n = (1-\delta)\delta^{n-1}$, gives $p_0\delta = \sum_{k=2}^{\infty} g_k \delta^k$, which is independent of n. As $p_0 = 1 - g_1$, the equation further simplifies to $\delta = \sum_{k=1}^{\infty} g_k \delta^k$, which has two roots. The strictly positive root gives the correct distribution.

[2] In fact, this is true for the G/M/1 queue, where random variable A denotes the interarrival time. Then $p_n = \int_0^{\infty} (\mu t)^n/n! \cdot e^{-\mu t}\, dA(t)$ and $0 < \delta < 1$ is the root of equation $x = A^*(\mu(1-x))$ [3]. Hence, the results in this section generalize to G/M/1 systems.

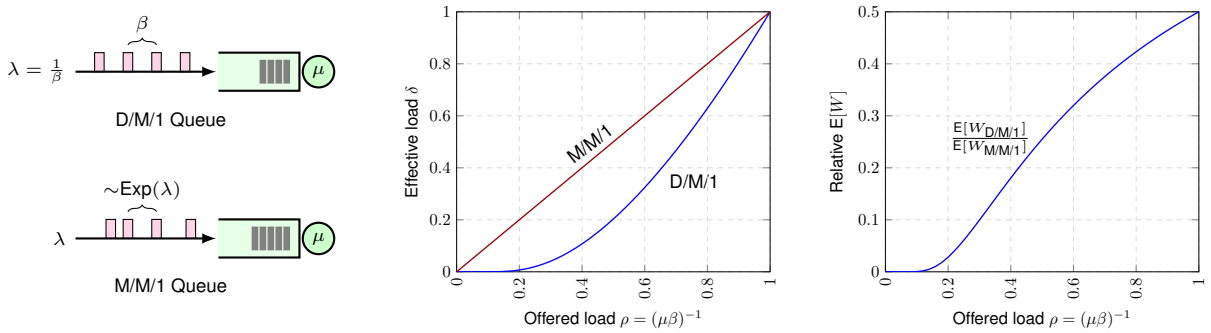

Fig. 1. D/M/1 vs. M/M/1: The left figure depicts the effective load δ, and the right figure the relative performance in terms of the mean waiting time.

service times with mean $1/\mu$, the (remaining) sojourn time of the n^{th} job in the FCFS queue is

$$T_n \sim X_1 + \dots X_n,$$

where $X_i \sim Exp(\mu)$ are i.i.d. random variables. The sum of exponential random variables obeys Erlang distribution, $T_n \sim Erlang(n, \mu)$, and

$$P\{T_n > \tau\} = e^{-\mu\tau}\left(1 + \mu\tau + \dots + \frac{(\mu\tau)^{n-1}}{(n-1)!}\right). \quad (6)$$

Defining the immediate cost $c(n)$ according to (6) *gives*

$$c_{t\tau}(n) = P\{T_{n+1} > \tau\}. \quad (7)$$

which corresponds to the (mean) deadline violation cost for a job arriving in state n.

Similarly, for the deadlines w.r.t. waiting time we have

$$c_{w\tau}(n) = P\{T_n > \tau\}. \quad (8)$$

Lemma 1. *The deadline violation probabilities in the D/M/1 queue are*

$$r_{t\tau} = P\{T > \tau\} = \quad e^{-\mu\tau(1-\delta)}, \quad (9)$$

$$r_{w\tau} = P\{W > \tau\} = \delta\, e^{-\mu\tau(1-\delta)}. \quad (10)$$

Proof. The number of jobs, N, in steady state obeys the geometric distribution, $N \sim \text{Geom}(\mu(1-\delta))$. The sojourn time T is a random sum,

$$T \sim X_1 + \dots X_N,$$

where the $X_i \sim \text{Exp}(\mu)$ are i.i.d., and as the *"geometric sum"* of exponentially distributed random variables is exponentially distributed, we have

$$T \sim \text{Exp}(\mu(1-\delta)),$$

yielding (9). For the deadline with respect to waiting time,

$$\begin{aligned}\text{P}\{W > \tau\} &= \sum_{n=1}^{\infty}(1-\delta)\delta^n e^{-\mu\tau}\left(1 + \mu\tau + \dots + \frac{(\mu/\tau)^{n-1}}{(n-1)!}\right)\\ &= \delta\sum_{n=0}^{\infty}(1-\delta)\delta^n e^{-\mu\tau}\left(1 + \mu\tau + \dots + \frac{(\mu/\tau)^{n}}{n!}\right)\end{aligned}$$

yielding $\text{P}\{W > \tau\} = \delta\,\text{P}\{T > \tau\}$. □

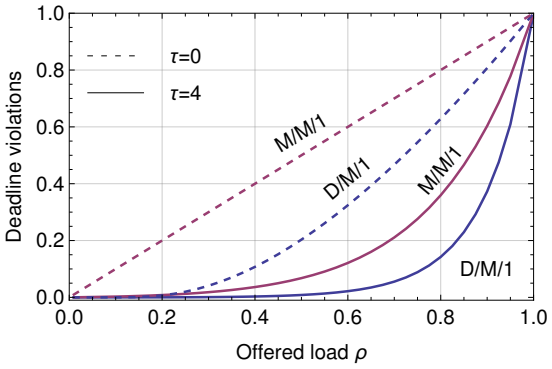

Fig. 2. Deadline violation probability for waiting time depicted for the D/M/1 and M/M/1 queues with $\mu = 1$ as a function of the offered load ρ. Dashed curves correspond to a very strict target deadline, $\tau = 0$, and the solid lines to a more reasonable target deadline, $\tau = 4$.

The above cost functions give some insight on the range of performance metrics one could easily define for this model. The corresponding mean costs give the steady-state performance for single-server systems.

Remark 1. *Note that δ of the D/M/1 queue is such that*

$$\delta = P\{T > \beta\} = 1 - P\{\textit{idle in a slot of length } \beta\}.$$

Similarly, with the M/M/1 queue, we also have

$$\rho = P\{T > A\},$$

where $A \sim Exp(\lambda)$ is an interarrvial time.

Example 3. *Let us compare the performance of the D/M/1 and M/M/1 queues in terms of the deadline violation metric for waiting time. We let $\mu = 1$ and then vary the offered load ρ. The deadline is assumed to be either very strict, $\tau = 0$, or a more reasonable one, $\tau = 4$. The results are depicted in Figure 2, which again highlight the benefits of having more regular inter-arrival times.*

It is worth noting that the above example illustrates also the performance of large parallel server systems subject to random and round-robin routing, as discussed later in Section III.

B. Value function for D/M/1

In contrast, a value function characterizes how "good" or "bad" some state is relative to the mean performance of the same system. In particular, it enables one policy improvement step, yielding a better (control) policy. Formally, the value function (without discounting) is defined as follows:

Definition 2 (Value function)**.** *The value function is the expected cost difference in the infinite time-horizon between a system initially in state n and a system in equilibrium,*

$$v_n \triangleq \lim_{t\to\infty} E[V_n(t) - rt], \tag{11}$$

where the random variable $V_n(t)$ denotes the total costs incurred during time $(0,t)$ when the system is initially in state n, and r is the mean cost rate.

In our case, state n denotes the number of jobs in the D/M/1 queue *immediately before an arrival*. Further, we assume that present jobs have *paid upon arrival* according to $c(n)$, e.g., according to their expected waiting time (see Example 1). Then a sufficient state description is the number of jobs in the system, $X_t = n$. The corresponding value function is referred to as the number-aware value function, for which it is straightforward to give a recursive expression by means of dynamic programming.

Theorem 1. *The value function for the number-aware D/M/1 queue with respect to arbitrary immediate costs $c(n)$ is given by the recursive function,*

$$v_{n+1} = (r - c(n) + v_n)\cdot e^{\mu\beta} - \sum_{j=1}^{n} \frac{(\mu\beta)^{n+1-j}}{(n+1-j)!}\cdot v_j, \tag{12}$$

for $n = 0, 1, 2, \ldots$, with the convention that $v_0 = 0$.

Proof. The value function v_n satisfies the dynamic programming equation (see Bellman [7], or Howard [8]),

$$v_n = \overbrace{c(n) - r}^{\text{one time step}} + \overbrace{\sum_{j=0}^{n} \mathrm{P}\{D = j\}\cdot v_{n+1-j} + \mathrm{P}\{D \geq n+1\}\cdot v_0}^{\text{mean difference in costs incurred in the future}},$$

for $n = 0, 1, 2, \ldots$, where r is the mean cost (rate), and D denotes the number of potential departures in one time step, $D \sim \text{Poisson}(\mu\beta)$. Solving for v_{n+1}, one obtains

$$v_{n+1} = (r - c(n) + v_n)\cdot e^{\mu\beta} - \sum_{j=1}^{n} \frac{(\mu\beta)^j}{j!}\cdot v_{n+1-j} - \left(e^{\mu\beta} - \sum_{j=0}^{n}\frac{(\mu\beta)^j}{j!}\right)\cdot v_0 \qquad n = 0, 1, 2, \ldots$$

As a constant term in the value function is immaterial, we can set $v_0 = 0$, yielding

$$v_{n+1} = (r - c(n) + v_n)\, e^{\mu\beta} - \sum_{j=1}^{n} \frac{(\mu\beta)^j}{j!}\, v_{n+1-j},$$

for $n = 0, 1, 2, \ldots$. Reversing the order in the summation then gives (12). □

Note that Eq. (12) expresses v_{n+1} as a function of $v_1, \ldots, v_n$, except for $n = 0$, which gives v_1 as a function of v_0 that we chose to set to zero, $v_0 = 0$. In principle, we can thus compute v_n for any (finite) n recursively.

Note also that the v_n's characterize the deviation from the mean cost rate from the time instant just prior the arrival. For the time instants right after the new job has joined the queue, we have $v^*_{n+1} = v_n - (c(n) + r)$. As the constant offset in value functions is immaterial, adding $c(0) - r$ gives us

$$v^*_{n+1} = v_n - (c(n) - c(0)), \qquad n = 0, 1, 2, \ldots \tag{13}$$

so that $v^*_1 = 0$. The state $n = 0$ after "an arrival" is transient. The corresponding value function is

$$v^*_0 = (c(0) - r) - v^*_1 = c(0) - r.$$

Remark 2 (M/M/1 Queue)**.** *Consider the M/M/1 queue with immediate costs $c(n)$. An equivalent recursive form for the corresponding value function (upon an arrival to state n) is*

$$v_{n+1} = (r - c(n) + v_n)\cdot \frac{1}{1-q} - \sum_{j=1}^{n} q^{n+1-j}\cdot v_j, \tag{14}$$

which follows from the observation that with the exponentially distributed inter-arrival times of the M/M/1 queue, the number of potential departures before the next arrival obeys the geometric distribution with parameter $q = \mu/(\mu+\lambda)$. For example, the immediate cost for sojourn time is given by (4)*, and the mean sojourn time is $r = (\mu - \lambda)^{-1}$, yielding*

$$v_n = \frac{n(2 + \rho + \rho n)}{2\mu(1-\rho)},$$

whereas for an arbitrary time instant (without an arrival), we would have

$$\tilde{v}_n = \frac{n(n+1)}{2\mu}\cdot\frac{\rho}{1-\rho},$$

which corresponds to the jobs arriving in future. Adding the expected sojourn time of the present n jobs, $n(n+1)/(2\mu)$, into the above then gives

$$v'_n = \frac{n(n+1)}{2(\mu-\lambda)},$$

which is the value function for a system where costs are incurred continuously at the rate equal to the number of jobs in the system [9], [10], [11].

Example 4. *As an example, the first four relative values of the D/M/1 queue with respect to the mean waiting time costs,* $c_w(n) = n/\mu$*, are*

$$\begin{aligned}
v_0 &= 0,\\
v_1 &= \frac{\delta}{1-\delta}\cdot\frac{e^{\mu\beta}}{\mu},\\
v_2 &= \left[(e^{\mu\beta}+1-\mu\beta)\cdot\frac{\delta}{1-\delta}-1\right]\cdot\frac{e^{\mu\beta}}{\mu},\\
v_3 &= \left[\left(e^{\mu\beta}\left(e^{\mu\beta}+1-2\mu\beta\right)+1-\mu\beta+\frac{(\mu\beta)^2}{2}\right)\frac{\delta}{1-\delta}\right.\\
&\qquad\left.-\left(e^{\mu\beta}+2-\mu\beta\right)\right]\cdot\frac{e^{\mu\beta}}{\mu},
\end{aligned}$$

where δ *is the root of* (1)*.*

Example 5. *The first few relative values with respect to deadline on response time, given by* (7)*, in the D/M/1 queue are*

$$\begin{aligned}
v_0 &= 0,\\
v_1 &= e^{\mu(\beta-\tau)}\left(e^{\delta\mu\tau}-1\right),\\
v_2 &= e^{\mu(\beta-\tau)}\left(e^{\mu(\beta+\delta\tau)}+(1-\beta\mu)e^{\delta\mu\tau}+\mu(\beta-\tau)\right.\\
&\qquad\left.-\ e^{\beta\mu}-1\right),\\
v_3 &= -\frac{1}{2}e^{\mu(\beta-\tau)}\left[(4\beta\mu-2)e^{\mu(\beta+\delta\tau)}-2e^{2\beta\mu+\delta\mu\tau}\right.\\
&\quad-(\beta\mu(\beta\mu-2)+2)e^{\delta\mu\tau}+e^{\beta\mu}(-4\beta\mu+2\mu\tau+2)\\
&\quad\left.+\mu(\beta-\tau)(\beta\mu-\mu\tau-2)+2e^{2\beta\mu}+2\right],
\end{aligned}$$

where δ *is the corresponding constant from the steady-state distribution.*

III. Applications

In this section, we demonstrate the possible applications of the new results.

A. Rescheduling appointments

The D/M/1 queue serves as a good model, when patients make appointments to a doctor, and time slots are available, e.g., once every 30 minutes. Hence, by design, we have full control over the arrival stream, whereas the service times remain random with a known distribution (exponential in this case). Stability requires that the mean service time be shorter than inter-arrival times. At the same time, economic interests and efficient use of server capacity (e.g., doctors) encourage us to operate as close to $\rho = 1$ as possible (through an appropriately chosen time interval β). However, the perceived quality of service improves as ρ is decreased. Consequently, in practice one strives to strike a balance betweeen conflicting objectives.

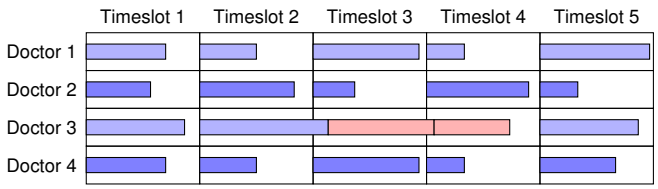

Fig. 3. Appointment system with 4 "servers".

A health clinic often has multiple doctors, which corresponds to **an appointment system with parallel servers**. For simplicity, we assume a fully booked system of n servers, where a batch of n (identical) jobs arrive at the start of every time slot. The graph in Figure 3 depicts a sample scenario with 4 servers (doctors in a health clinic). Patients have booked appointments and the schedule is full. In most cases, the visit ends in time and the next patient gets in at the scheduled time, $W = 0$. However, the 2nd patient of Doctor 3 spends more than the scheduled time, which causes an extra waiting time for the following two patients ($W > 0$). If the backlog for one doctor increases too much, it is probably useful to direct one of her/his patient to some other doctor. The so-called *rescheduling policy*, denoted by α makes such decisions.

If jobs and servers are identical, the obvious solution is to balance the load by using the Join-the-Shortest-Queue (JSQ) [12]. However, when servers are heterogeneous in terms of speed or cost, or if jobs, e.g., have different holding costs, it may be beneficial to direct more jobs to some server(s) than to others. Here we assume that the patients dislike being transferred to a doctor other than the one they booked, so we introduce a fixed extra penalty cost e for each such action.

The default policy α_0 thus assigns each job to its respective server (doctor) that was originally scheduled for them. In this case, the system decomposes into n parallel and independent D/M/1 queues, and the queues can be analyzed separately. In particular, the value function of the system is the sum of the queue-specific value functions.[3]

A dynamic policy would then occasionally deviate from the default action. We utilize the value functions to this end.

1) Admission costs: Now we are ready to determine near optimal rescheduling actions based on the value functions discussed in the previous section.

Let queue q be in state n (it has n jobs just before the time of the batch arrival), and suppose that in this decision round we consider assigning k new jobs to it. Let v_n^* denote the relative value when the system is observed *immediately after* a job has arrived, as in (13). These two value functions have a simple relationship,

$$v_n = c(n) - c(0) + v_{n+1}^*, \qquad n = 0, 1, 2, \ldots$$

where the constant offsets are chosen so that $v_0 = v_1^* = 0$. With these, the admission cost of k jobs to queue q is

$$a_n(k) = c(n) + \ldots + c(n+k-1) + v_{n+k}^* - v_n^*,$$

[3]All this is in analogy with the more common setting where the arrival process is Poisson.

i.e., the sum of the costs the k new jobs incur and the penalty imposed on later arriving jobs, characterized by the corresponding value function. As was the case for the value function, a constant offset can be subtracted from admission costs. In the expression above, the term $-v_n^*$ corresponds to this (it is independent of the scheduling action assigning the n jobs to n servers). Here the baseline is thus chosen so that *not reassigning the default patient* has zero cost. The additional annoyance cost of reassigning $k-1$ jobs to server q, in addition to its originally assigned job, is

$$a_n(k) = (k-1)^+ \cdot e,$$

where $(x)^+ = \max\{0, x\}$ and e is a free parameter.

Example 6. *With the waiting time metric, $c(n) = n/\mu$, the admission cost for k new jobs is*

$$a_n(k) = \frac{n + (n+1) + \ldots + (n+k-1)}{\mu} + v_{n+k}^* - v_n^*,$$

which reduces to

$$a_n(k) = \frac{k(2n+k-1)}{2\mu} + v_{n+k}^* - v_n^*.$$

In particular, assigning no jobs has zero cost, $a_n(0) = 0$, and the cost of assigning a single job is

$$a_n(1) = \frac{n}{\mu} + v_{n+1}^* - v_n^*,$$

so that the first few admission costs for waiting time are as follows:

n	$a_n(1)$
0	r
1	$r\, e^{\mu\beta}$
2	$e^{\mu\beta}(e^{\mu\beta} - \mu\beta)\, r - (e^{\mu\beta} - 1)\frac{1}{\mu}$
3	$e^{\beta\mu}\left(e^{\beta\mu}(e^{\beta\mu} - 2\beta\mu) + \frac{1}{2}(\beta\mu)^2\right) r$ $- \left(e^{\beta\mu}\left(e^{\beta\mu} - \beta\mu + 1\right) - 2\right)\frac{1}{\mu}$

Even though the expressions become lengthy, numerical values are easy to compute using the recursive expression.

Note that it is possible to generalize the model and our results to include an option to *reject* or *reschedule* jobs. Such an action could have a fixed or job-specific cost. More generally, we might have a rescheduling cost that is decreasing in the advance warning time. That is, at the start of a slot, given the current number, we might want to cancel/reschedule the arrival in the next slot. We leave such extensions to the reader.

B. Dispatching system

Our next application is a dispatching system, in which jobs arrive individually at constant time-intervals β and are routed immediately upon arrival to one of the k available servers. This is illustrated in Figure 4. The service time at server i is exponentially distributed with rate μ_i, i.e., we allow heterogeneous servers. With identical servers and identical jobs, JSQ is often the optimal policy. However, if jobs have different *holding costs* or (some) jobs have deadlines, the situation is more interesting and policy iteration is likely to give a better policy than JSQ.

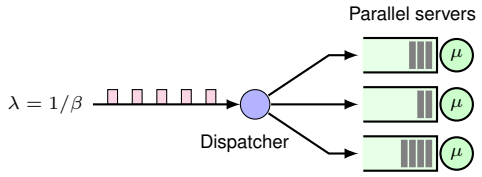

Fig. 4. Model for a parallel server system, where the inter-arrival time between jobs is constant β and service times are independent and exponentially distributed with mean $1/\mu$.

The fact that the arrival process to every server has a "memory" complicates the situation somewhat. Suppose our (basic) policy is round-robin:

Definition 3 (Round-robin)**.** *The round-robin (RR) dispatching policy assigns jobs sequentially to servers in a fixed order:* $1, 2, \ldots, k, 1, 2, \ldots$.

As a result, the arrival process to every server has deterministic time intervals βn, and the system decomposes similarly as with the rescheduling problem in the previous section. Round-robin is known to be the optimal policy in many settings, when the available information is basically the past decisions (and that the system was initially, e.g., empty) [13], [14].

However, the moment we deviate from the basic routing pattern and assign a job to server j instead of server i, the situation becomes more complex. Namely,

- At every decision point, all servers except one are now in the middle of their respective arrival cycles (phases) with the remaining time being $i\beta$, $i = 1, \ldots, (k-1)$.
- However, our value functions hold for the time instants corresponding to the arrival ($i = 0$).
- Thefore, we will need to compute the expected value for each server after the current decision.
- In order to get "the best" understanding about the current state and the cost of respective actions, one should consider all possible orders of the phases (of RR). This corresponds to $k!$ different permutations of the sequence $\{0, 1, \ldots, (k-1)\}$, which can be a large number.
- However, intuitively, a good heuristic rule for RR with identical servers is to order the queues in the increasing order of their queue lengths. Consequently, when k is large and a faster algorithm is needed, the search space can be pruned, e.g., by considering only the sequences where the first server is varied and the rest are ordered according to their queue lengths (correspondingly, we would have k sequences instead of $k!$).

Thus, for an *informed decision*, we need to compute the expected value of each server at these intermediate time instants based on the number of jobs currently in them. Given the current state of a server is n and the next job will exceptionally arrive after time $t = i\beta$, instead of immediately ($t = 0$), we proceed similarly as we did in the proof of

Algorithm 1 FPI-based dispatching policy

function FPI($n_1, \ldots, n_k$) ▷ Jobs in each server
  $a \leftarrow 0$
  for π in all permutations of $\{1, 2, \ldots, k\}$ **do**
    $w \leftarrow 0$
    for $i = 1, \ldots, k$ **do**
      $j = \pi(i)$ ▷ Server for the ith job
      $w \leftarrow w + v_{n_j}^{(j)}((i-1)\beta)$ ▷ Add value function
    end for
    if $a = 0$ or $w_{\min} > w$ **then**
      $a = \pi(1)$ ▷ Action for this job
      $w_{\min} = w$
    end if
  end for
  return a ▷ Action with the lowest relative value
end function

Theorem 1:

$$v_n^*(t) = \sum_{j=0}^{n-1} \mathrm{P}\{D_t = j\} \cdot v_{n-j} + \mathrm{P}\{D_t \geq n\} \cdot v_0, \qquad (15)$$

where $t \geq 0$. The above expression gives the expected value of the queue given its current state is n and the next job is bound to arrive after time t. As before, the random variable D_t corresponds to the number of possible departures, $D_t \sim \text{Poisson}(\mu t)$.

When jobs are not identical and a job's type, or class, becomes known upon arrival (before the dispatching decision), such information should be taken into account when evaluating different actions. This affects only the queue receiving the current job, and the refined value function for it is ($i = 0$)

$$v_n^* = c^*(n) - c(n) + v_n, \qquad (16)$$

where $c^*(n)$ gives the (expected) cost of the given new job (i.e., conditioned on the information that got revealed upon job's arrival), and $c(n)$ is the corresponding cost on average (over all jobs).

Definition 4 (Policy iteration)**.** *The policy iteration based dispatching policy routes a new job to the server that minimizes the expected future costs obtained using* (15) *(for* $i > 0$*) and* (16) *(for* $i = 0$*). We call this policy FPI for First Policy Iteration (step).*

The concrete steps involved are described in Algorithm 1, which evaluates the value function for all possible ways to order the servers given the RR policy is used for this and future jobs. The order yielding the smallest (expected) cost is chosen. The algorithm assumes that queue-specific value functions $v_n^{(i)}(t)$ for queue i are available, which in turn depend on the chosen cost function and basic policy. In practice, one is likely to compute the $v_n^{(i)}$, i.e., the value functions for time instants upon arrival, and then utilize (15) on the fly to compute the necessary value functions for states in between the arrivals.

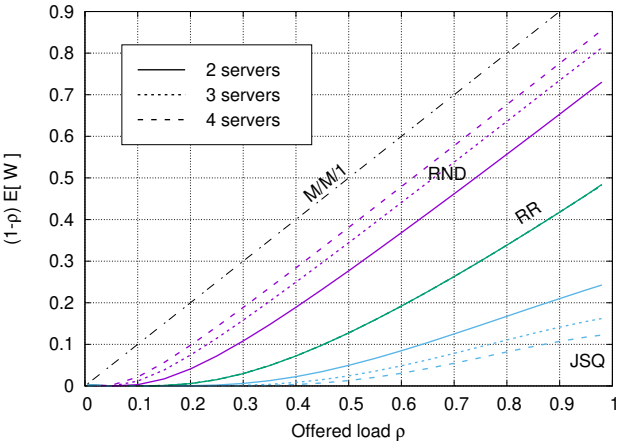

Fig. 5. Mean (scaled) waiting time with 2, 3 and 4 identical servers. FPI is omitted as it yields JSQ in this case. RR gives the same result with any number of servers as the arrival process to each server remains the same.

IV. Numerical examples

We consider the following heuristic policies:

1) Random split (RND) which assigns jobs independently to k servers with probabilities $p_i \propto \mu_i$, where μ_i is the service rate of server i, where $i = 1, \ldots, k$.
2) Round-robin assigns jobs sequentially to servers. By default, the sequence is $1, 2, \ldots, k, 1, 2, \ldots$.
3) Join-the-Shortest-Queue (JSQ), assigning jobs to the server with the least number of jobs,
$$\alpha_{\text{JSQ}} = \underset{i}{\text{argmin}} \ \{n_i\},$$
where n_i denotes the number of jobs in server i.
4) Shortest-Expected-Delay (SED) takes the service rates into account and chooses the server according to
$$\alpha_{\text{SED}} = \underset{i}{\text{argmin}} \ \{n_i/\mu_i\}.$$

With JSQ and SED, ties are resolved in favor of the server with a smaller index. JSQ is often the optimal policy when both servers and jobs are identical (see, e.g., [12], [13]). Note that all policies are expected to be stable when $\rho < 1$.

A. Identical Parallel Servers

First we assume a system that consists of identical servers with rates μ. Figure 5 depicts the mean waiting time, scaled by $1-\rho$, as a function of ρ. The solid lines correspond to the two server systems. Basically, we can make two observations. First, the results quantify benefits from having more state information available (RR knows the past decisions, JSQ the current number of jobs in each server). Second, the performance with RND gets worse as the number of servers increases, whereas with JSQ the situation improves (and JSQ is equivalent to SED in this case). In fact, it turns out that in a large system, as $k \to \infty$, RND yields k independent M/M/1 queues, whereas JSQ finds an idle server for every new job (in the limit when $k \to \infty$).

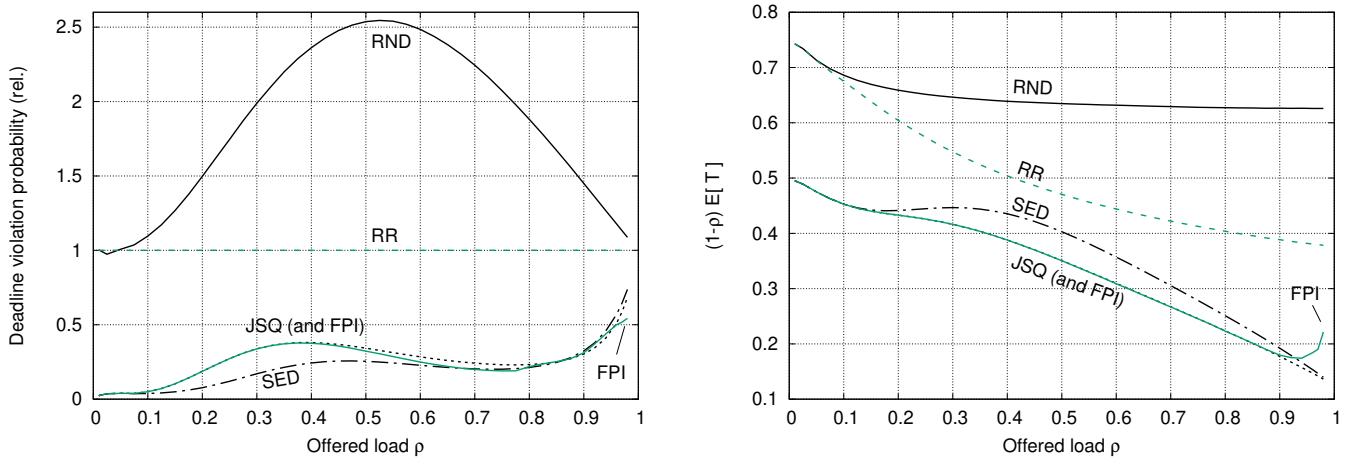

Fig. 6. The relative deadline violation probability (left) and the (scaled) mean waiting time (right) with heterogeneous service rates. RR is based on the load balancing sequence $1, 2, 1, 3, 1, 2, \ldots$.

B. Heterogeneous Parallel Servers

The situation becomes more interesting when servers are not identical. Thus, next we assume a system consisting of three heterogeneous servers:

- Server 1 with service rate 2μ,
- Server 2 with service rate μ,
- Server 3 with service rate μ.

The primary Server 1 is twice as fast as the two secondary servers. In this case, the constant inter-arrival times can be preserved by a round-robin type of sequence $1, 2, 1, 3, 1, 2, \ldots$, which assigns every other job to Server 1, and the rest alternatively to Server 2 and 3. We let α_0 denote this policy.

In addition to RR, we evaluate three other heuristic policies, RND, JSQ and SED, and FPI based on RR. As for the performance metrics, we consider the following two:

1) Mean response time $\mathrm{E}[T]$
2) Deadline violation probability for response time with $\tau = 4/\mu$, so that the risk of deadline violation with the slower servers is of some concern even if a server is currently idle.

Figure 6 depicts the simulation results in the heterogeneous scenario with respect to the deadline on response time at $\tau = 4/\mu$ (left) and the (scaled) mean response time (right). RR offers a significantly better performance than RND, but as in the case of identical servers, it loses to JSQ and other dynamic policies (SED and FPI). Note that FPI is based on RR.

C. Non-identical Jobs with Heterogeneous Servers

In our final numerical experiment, each job is associated with a job-specific holding cost, which are assumed to be independent uniformly distributed random variables, $H \sim \mathrm{U}(0, 2)$. Each job then incurs costs at the given rate during its whole sojourn time, and the task is to minimize the product $\mathrm{E}[HT]$. Otherwise, we assume the same heterogeneous system of three parallel servers, and apply the same set of heuristic policies. The numerical results are depicted in Figure 7. In this case, we have chosen SED as the reference point. Note

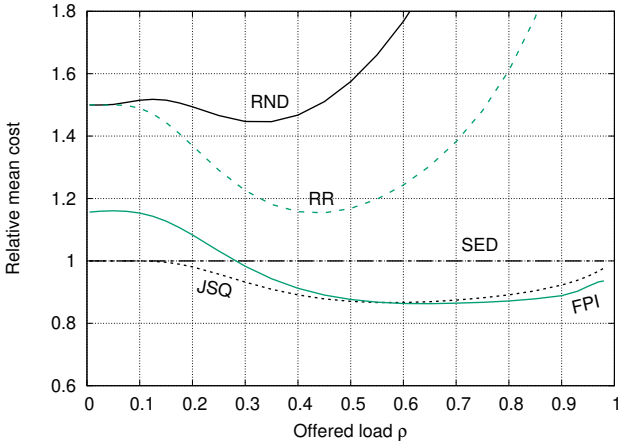

Fig. 7. Relative costs with three heterogeneous servers when jobs have varying holding costs. When ρ is very small, the heuristic JSQ and SED policies work well, but as $\rho > 0.6$, the FPI policy offers the best performance.

that SED corresponds to greedy selfish jobs minimizing their individual costs. Interestingly, SED tends to have a clearly worse performance than JSQ. FPI, being an adaptive cost-aware policy, adapts to the new cost structure and has the best performance when the load is medium or high (around 0.6 or higher).

Similarly, the FPI approach can be applied to scenarios where processing a job has a different cost at each server. For example, extra servers (virtual machines) could be available for rental at some higher cost.

V. Conclusions

In this paper, we analyzed the number-aware D/M/1 queue, where the state information is the number of jobs. The arrival process is deterministic and jobs are assumed to arrive at fixed time intervals. Such scenarios arise, e.g., in booking systems and when the system has control over arrivals. We studied the D/M/1 queue in the context of Markov decision processes. First, we gave a recursive formula to compute the

value function with respect to an arbitrary cost function. As example cases, we considered the mean waiting and response time, as well as, deadlines on waiting and response time.

The availability of the value function allows one to develop sound control strategies for the related systems. For example, we can reschedule appointments in a health clinic (depending on the objective functions), or we can develop efficient cost-aware policies for dispatching jobs to parallel servers. In contrast to past work, the assumed arrival process has memory, which complicates its application to the dispatching problem. However, as demonstrated, this can be worked out by an appropriate conditioning, which resembles the analysis of Erl/G/1 queues [1] and the lookahead idea proposed in [15]. In our numerical examples, the resulting FPI policies performed on par with the dynamic JSQ and SED policies. The shortcoming of JSQ and SED is that they are blind to cost structures and job-specific attributes such as high and low priority jobs having, e.g., different deadlines and/or cost parameters. In contrast, the FPI approach extends to all such cases and, in this sense, it is superior to the aforementioned heuristics.

Acknowledgements

This work was supported by the University of Iceland Research Fund in the RL-STAR project.

References

[1] E. Hyytiä and S. Aalto, "On round-robin routing policy with FCFS and LCFS scheduling," *Performance Evaluation*, vol. 97, pp. 83–103, Mar. 2016.

[2] B. Jansson, "Choosing a good appointment system-a study of queues of the type (D, M, 1)," *Operations Research*, vol. 14, no. 2, pp. 292–312, 1966.

[3] L. Kleinrock, *Queueing Systems, Volume I: Theory*. Wiley Interscience, 1975.

[4] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.

[5] E. Hyytiä, R. Righter, and J. Virtamo, "Meeting soft deadlines in single- and multi-server systems," in *28th International Teletraffic Congress (ITC'28)*, Würzburg, Germany, Sep. 2016.

[6] E. Hyytiä, R. Righter, O. Bilenne, and X. Wu, "Dispatching discrete-size jobs with multiple deadlines to parallel heterogeneous servers," in *Systems modeling: methodologies and tools*, ser. EAI/Springer Innovations in Communications and Computing, A. Puliafito and K. Trivedi, Eds. Springer, 2018, pp. 29–46.

[7] R. Bellman, *Dynamic programming*. Princeton University Press, 1957.

[8] R. A. Howard, *Dynamic Probabilistic Systems, Volume II: Semi-Markov and Decision Processes*. Wiley Interscience, 1971.

[9] K. R. Krishnan, "Joining the right queue: a Markov decision rule," in *Proc. of the 28th Conference on Decision and Control*, Dec. 1987, pp. 1863–1868.

[10] S. Aalto and J. Virtamo, "Basic packet routing problem," in *The thirteenth Nordic teletraffic seminar NTS-13*, Trondheim, Norway, Aug. 1996, pp. 85–97.

[11] P. Whittle, *Optimal Control: Basics and Beyond*. Wiley, 1996.

[12] W. Winston, "Optimality of the shortest line discipline," *Journal of Applied Probability*, vol. 14, pp. 181–189, 1977.

[13] A. Ephremides, P. Varaiya, and J. Walrand, "A simple dynamic routing problem," *IEEE Transactions on Automatic Control*, vol. 25, no. 4, pp. 690–693, Aug. 1980.

[14] Z. Liu and R. Righter, "Optimal load balancing on distributed homogeneous unreliable processors," *Operations Research*, vol. 46, no. 4, pp. 563–573, 1998.

[15] E. Hyytiä, "Lookahead actions in dispatching to parallel queues," *Performance Evaluation*, vol. 70, no. 10, pp. 859–872, 2013.