Predicting Latency Distributions of Aperiodic Time-Critical Services

Haoran Li, Chenyang Lu, Christopher Gill
Cyber-Physical Systems Laboratory, Washington University in St. Louis
{lihaoran, lu, cdgill}@wustl.edu

Abstract—**There is increasing interest in supporting time-critical services in cloud computing environments. Those cloud services differ from traditional hard real-time systems in three aspects. First, cloud services usually involve latency requirements in terms of probabilistic tail latency instead of hard deadlines. Second, some cloud services need to handle aperiodic requests for stochastic arrival processes instead of traditional periodic or sporadic models. Finally, the computing platform must provide performance isolation between time-critical services and other workloads. It is therefore essential to provision resources to meet different tail latency requirements. As a step towards cloud services with stochastic latency guarantees, this paper presents a stochastic response time analysis for aperiodic services following a Poisson arrival process on computing platforms that schedue time-critical services as deferrable servers. The stochastic analysis enables a service operator to provision CPU resources for aperiodic services to achieve a desired tail latency. We evaluated the method in two case studies, one involving a synthetic service and another involving a Redis service, both on a testbed based on Xen 4.10. The results demonstrate the validity and efficacy of our method in a practical setting.**

Index Terms—**Time-sensitive Service, Deferrable Server, Queueing Model, Response Time Ananlysis, Tail Latency**

I. Introduction

Time-critical services are increasingly hosted in the cloud to take advantage of the flexibility and scalability of cloud computing platforms. Examples include streaming analytics [1], interactive services [2], and in-memory data stores [3]. Cloud providers face a tremendous challenge to meet the latency requirements of time-critical services. In contrast to traditional real-time systems, the service-level objective (SLO) of a time-critical service is usually in terms of a target *tail latency* instead of hard deadlines. Moreover, requests for cloud services may arrive *aperiodically* following stochastic arrival processes that depart from periodic or sporadic task models. Finally, it is essential to enforce *performance isolation* between cloud services in a shared, multi-tenant environment. A service (or micro-service [4]) may be instantiated at different granularities, ranging from a process, to a container, to a virtual machine. For CPU resources, which are the focus of this paper, a common approach to performance isolation is to schedule a service as a *scheduling server* that is allowed to execute for a specified *budget* within each *period*. As examples, Quest-V [5] uses the PIBS server for micro-service processes, whereas a Linux process, when scheduled by `SCHED_DEADLINE` [6], is governed by a constant bandwidth server (CBS), and a virtual CPU (VCPU) is scheduled as a deferrable server (DS) when scheduled by the Xen RTDS [7], [8] scheduler.

The confluence of stochastic latency requirements, aperiodic arrivals, and performance isolation mechanisms makes it highly challenging to analyze the response time of a time-critical service. The stochastic arrivals and the resultant queueing delays make response time analysis non-trivial, even for the highest-priority task in fixed-priority scheduling. The enforcement of CPU budgets for performance isolation further aggravates the complexity of the analysis.

Traditionally, response time analysis in real-time systems focuses on bounding the worst-case response time. This approach does not work for time-critical services with aperiodic stochastic arrivals and tail latency requirements. Due to the stochastic arrivals of service requests, the worst-case response time cannot be bounded. While soft real-time scheduling approaches are geared towards achieving bounded tardiness [9]–[11], they focus on periodic tasks. Similarly, earlier efforts on stochastic analysis have been proposed for periodic tasks [12]–[17]. Moreover, existing aperiodic scheduling approaches based on scheduling servers [18]–[21] and aperiodic utilization bounds [22] are not designed to provide offline guarantees for aperiodic tasks. Hierarchical and compositional scheduling analysis [23]–[26] leverages scheduling servers to achieve performance isolation, but the analysis approach is generally designed for periodic tasks and deadlines instead of tail latency.

Queueing theory provides tools to derive the response time distribution subject to stochastic arrivals. Real-time calculus [27], [28] and real-time queueing theory [29], [30] extend probabilistic queueing theory to derive hard upper and lower performance bounds instead of tail latency. Furthermore, while traditional queueing theory usually models an always-on server, a scheduling server algorithm (e.g., deferrable server) dictates that the server is no longer always active, i.e., the server is suspended when it runs out of its budget. It is therefore necessary to extend queueing theory to model and analyze scheduling servers.

This paper takes a first step towards response time analysis for time-critical services. We consider a time-critical service scheduled as a deferrable server for performance isolation, and develop a numerical method for computing the stationary[1]

[1]A stationary distribution of a Markov chain is a probabilistic distribution that remains unchanged in the Markov chain as time progresses [31].

response time distribution of a stochastic Poisson arrival process. From a queueing theory perspective, we denote the system as M/D(DS)/1. This notation extends the established M/D/1 queue (in Kendall's notation [32][2]), to highlight that the server model is changed from an always-active server to a deferrable server (DS).

We first observe that a scheduling server in real-time scheduling theory can be modeled as a periodic service queueing model that has been studied in the queueing theory literature [33]–[37]. Combining scheduling analysis and virtual waiting time analysis, we then establish a transformation between periodic and deferrable servers. Finally, we derive an efficient method to compute the stationary response time distributions of M/D(DS)/1 models. We demonstrate how the response time distribution of an M/D(DS)/1 system enables us to configure a deferrable server to achieve the desired tail latency for a time-critical service. We implement our approach in a virtualization platform based on Xen 4.10 and evaluate two case studies, one involving a synthetic service and another involving Redis, a common in-memory data storage service. The results of these studies demonstrate the validity and efficacy of our numerical approach in supporting time-critical services in a practical setting.

II. Background

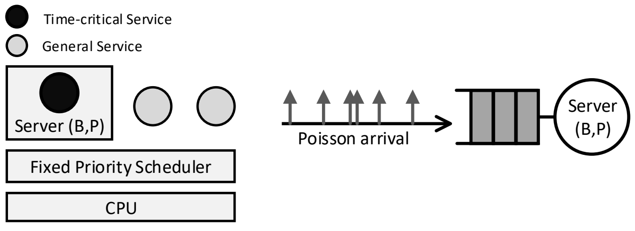
In queueing theory, an M/D/1 queue represents a single server queueing system, where job arrivals are determined by a Poisson process and the job service durations of a given task have the same constant value, i.e., they are deterministic. This queueing model, first studied and published by Erlang in 1909 [38], initiated the study of queueing theory.

With the development of an ever growing range of communication and computing systems, queueing theory has flourished and has been used both as a **predictive tool**, allowing one to predict the performance of a given system, and as a **design tool**, supporting system configurations to minimize response time. Numerous models involving different arrival patterns, service duration distributions, and multiple servers with different scheduling policies have been studied for different real-world scenarios.

A not-always-active server is of particular interest, and a queueing system with periodic service is a typical case. For example, a fixed-cycle traffic light that controls an intersection can be modeled as a periodic service for each lane; similarly, a TDMA network can provide several slots for each channel in each period.

While queueing systems with periodic service have received significant research attention, the rapid advancement of cloud technology and new system design paradigms (e.g., micro-services) require even more sophisticated service models. The deferrable server model has been relatively ignored by queueing theory, despite a recent surge in its real-world realizations: e.g., the RTDS scheduler of Xen [8], and vMPCP [39].

[2]In Kendall's Notation, "M" is for Poisson arrival, indicating the feature of "Memoryless"; "D" is for constant service time, i.e., "Deterministic".



(a) Scheduling Model (b) Queueing Model

Fig. 1: M/D(DS)/1 and M/D(PS)/1 Queueing Model

Predicting even the simplest queueing system involving a deferrable server policy is still demanding. In this paper, by establishing and analyzing the M/D(DS)/1 model, we provide a numerical method for computing the relevant stationary response time distribution. As a result, this method can act as both a predictive tool and a design tool.

III. System Model

Our goal is to predict the stationary response distribution of a time-critical service with a Poisson arrival process and a deterministic service time, when the service is governed by a deferrable server. As the first step towards establishing a scheduling theory for time-critical services, in this paper we investigate a relatively simple, but still practical, system model. Specifically, the system has M CPUs. There is one aperiodic, time-critical service assigned on each CPU. There may be other general services sharing the CPU with the time-critical service. The requests for the same time-critical service arrive following a Poisson process and are scheduled in a FIFO fashion. The system employs partitioned fixed-priority scheduling. The time-critical service is scheduled as a deferrable server assigned a higher priority than the other services on the same CPU. Fig. 1a shows a single CPU as an example. Despite its simplicity, this architecture is suitable for cloud or edge computing scenarios that provide a mix of time-critical and general services. We focus on CPU resources and do not consider inter-service interference arising from other shared resources, such as cache, memory, and I/O. Handling other resources will be part of future work.

Finding the stationary response time distribution of such a time-critical service can help us to validate whether the service level objective (e.g., a 90^{th} percentile latency of less than 10 ms) can be met, in order to configure practical server parameter settings and thus to provide desired performance.

Due to the stochastic nature of the arriving jobs, the system is amenable to analysis based on a probabilistic queueing model. We transfer the scheduling model (Fig. 1a) to the queueing model (Fig. 1b) for the purpose of analyzing the stochastic response time of the time-critical service. In the following section, we further introduce our queueing model, starting with the canonical M/D/1 queue.

M/D/1 Queue. An M/D/1 queue presents a single always-active server with Poisson arrivals and deterministic service times. This model has two parameters (λ, d). The *arrival rate*

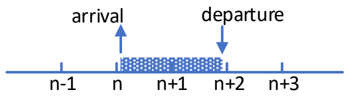


Fig. 2: Job Arrival and Departure

λ means that jobs' inter-arrival times follow an exponential distribution of parameter λ; the constant *service time* is d.

Periodic Server: M/D(PS)/1 Queue. Unlike the M/D/1 queue, a single periodic server is not always active. We use two additional parameters, *budget* and *period*, noted as (B, P), where $B \leq P$, to represent the periodic server model. Since the server has the highest priority, the server's schedule is fixed and switches between the on-state and off-state. When scheduling with other general tasks, as long as the periodic server is in the on-state, its budget linearly decreases, regardless whether it is processing a request. The server stays in the off-state (not providing service) during time $t \in [kP, kP + (P - B))$, and provides service in the on-state during time $t \in [kP + (P - B), (k + 1)P)$. The server runs jobs only when it is in the on-state.

Deferrable Server: M/D(DS)/1 Queue. A server in an M/D(DS)/1 queue is another not-always-active server: a deferrable server. Although M/D(DS)/1 can be also modeled by four parameters (λ, d, B, P) (like M/D(PS)/1, in Fig. 1b), the server policy is different. At the beginning of each period, the server replenishes its remaining budget to B. Whenever a job arrives and the server has remaining budget, it can run the job by consuming its budget. The server retains its remaining budget if it is not running, and if the budget is exhausted (reaches 0), the server stops providing any service until the next period, when the budget gets replenished.

System Stability. For a deferrable or periodic server with parameters (B, P), we define the *bandwidth* $W = \frac{B}{P}$. For an M/D/ arrival with parameters (λ, d), the system *utilization* is $U = \lambda d$. In order to make a system stable, say, to achieve statistical equilibrium, the system utilization U should be less than the server bandwidth B. Specifically, for M/D(PS)/1 and M/D(DS)/1 systems with parameters (λ, d, B, P), the stability condition is

$$U = \lambda d < \frac{B}{P} = W. \tag{1}$$

We are interested in the response time (sojourn time) distribution of an M/D(DS)/1 queue. In this paper, we present a numerical algorithm to derive the stationary response time distribution of a Poisson arrival process with a deterministic service time, subject to the deferrable server policy.

Challenge. The M/D(PS)/1 model has been studied in queueing theory (e.g., traffic flow under a periodic traffic signal) since the mid 1900s. However, it remains difficult to derive the explicit analytical result for the stationary response time distribution of M/D(PS)/1 queueing models. Because the M/D(DS)/1 model is much more complicated than the M/D(PS)/1 model, and since budget management for M/D(DS)/1 is correlated with stochastic arrivals, it is extremely challenging to derive an explicit analytical result for the stationary response time distribution for an M/D(DS)/1 model.

Thus, instead of studying the M/D(DS)/1 model in continuous time using mathematical tools like the *Laplace-Stieltjes Transform*, we use a discrete time analysis to approximate the continuous time M/D(DS)/1 model, as detailed in the next section. Thanks to the *Poisson Limit Theorem* [40], we can first study Bernoulli arrivals with deterministic service times, subject to the deferrable server policy (B/D(DS)/1 queue). Then, by choosing a small enough time quantum, we can closely approximate the M/D(DS)/1 model.

B/D(PS)/1 and B/D(DS)/1. Consider a single queue system where we divide the time axis into intervals of equal length. Each interval is a *slot*. Jobs arrive according to a Bernoulli process with parameter η: in each slot, either we have exactly one job arrival, with probability η, or we have no job arrival, with probability $1-\eta$. The inter-arrival time distribution of the Bernoulli process is a geometric distribution with parameter η. The service times of jobs are deterministic, denoted as d. A B/D(DS)/1 or B/D(PS)/1 queue can be modeled by the parameters (η, d, B, P).

Job arrival, service starting, and job departure (service completion) occur only at slot boundaries. For convenience, we assume that an job arrival ori the start of a service always occurs *just after* the slot boundary, and a job departs only *just before* the slot boundary. For example, in Fig. 2, a job arrives at n_+ and departs at $(n+2)_-$, starting its service just after the beginning of n-th slot, running for 2 time units, and departing just before the $(n + 2)$-th slot.

The server renders service according to the budget management policy, i.e., deferrable or periodic server policy. The stability condition is $U = \eta d < \frac{B}{P} = W$.

IV. Theoretical Properties and Algorithms

In this section, we compute the stationary response time distribution of an M/D(DS)/1 queue with the parameters (λ, d', B', P'). This M/D(DS)/1 queue can be approximated by a discrete B/D(DS)/1 queue with the parameters (η, d, B, P). We first illustrate how to quantize the system and map the parameters. We then study two queues, B/D/(PS)/1 and B/D(DS)/1, and focus on their *virtual waiting time process*. If a new job happens to arrive at the queueing system at time n, the server may still have pending jobs to render; and the server has to stay in active-state for $v[n]$ time units before handling this new job. This time $v[n]$ is defined as the *virtual waiting time*.

Assuming their virtual waiting time processes (discrete stochastic processes) are $\tilde{V}_{DS}[n]$ and $\tilde{V}_{PS}[n]$, we study the virtual waiting time at the start of each period, i.e., $V_{DS}[n] = \tilde{V}_{DS}[nP]$ and $V_{PS}[n] = \tilde{V}_{PS}[nP]$. $V_{DS}[n]$ and $V_{PS}[n]$ are Markov chains. We prove an equivalence in Theorem 1 and Corollary 5: if the system is stable, the two queues always have the same virtual waiting time distribution at the start of each server period, i.e., $V_{DS}[n] = V_{PS}[n]$.

Theorem 1 indicates that if we can compute the stationary virtual waiting time distribution at the start of the server

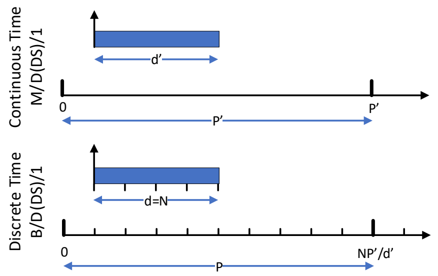


Fig. 3: Quantization: From Continuous Time to Discrete Time

Parameter	M/D(DS)/1	B/D(DS)/1
Arrival/Succeed Rate	λ	$\eta = \lambda d'/N$
Service Duration	d'	$d = N$
Period	P'	$P = NP'/d'$
Budget	B'	$B = NB'/d'$

TABLE I: Parameter Mapping

period, then for B/D(PS)/1, the result can be applied directly to the comparable B/D(DS)/1 model. Algorithm 1 leverages this feature and uses the B/D(PS)/1 model to compute the stationary virtual waiting time distribution at the start of the server period. Then, the stationary virtual waiting time distribution of B/D(DS)/1 at all slots within a server period can be computed recursively via Algorithm 2.

Finally, if the virtual waiting time of the queueing system upon one job's arrival is known, then the response time is deterministic, due to the FIFO policy and deterministic service time. Using the discrete version of the PASTA Theorem [31][3], in which "upon arrival at a station, a job observes the system as if in steady state at an arbitrary instant for the system without that job" [41], we can determine the stationary response time distribution, F_{DS}, for a B/D(DS)/1 queue.

A. Approximating the continuous M/D(DS)/1 model

As we discussed previously, we can employ a B/D(DS)/1 queue, using the Poisson Limit Theorem [40], to approximate the stationary response time distribution of an M/D(DS)/1 queue. Given an M/D(DS)/1 queue with parameters of (λ, d', P', B'), we can quantify the continuous time model by choosing a quantum (slot width) of d'/N and get a discrete time version, as shown in Fig. 3. Then, we can represent a correlated B/D(DS)/1 queue with the parameters (η, d, P, B) in this discrete time model. Using the invariance of utilization (i.e., $\lambda d' = \eta N$) and bandwidth (i.e., $\frac{B'}{P'} = \frac{B}{P}$), we can easily produce the mapping shown in Table I. Using the Poisson Limit Theorem [40], we can approximate a Poisson process by a Bernoulli process, given large enough N. As a result, we can approximate the M/D(DS)/1 system by mapping it to a B/D(DS)/1 system with an appropriate value of N.

[3]The discrete version is also called the BASTA Theorem – Bernoulli Arrivals See Time Average.

B. Virtual Waiting Time Equivalence at Start of Server Period

In this section, we show the equivalence of the virtual waiting times at the start of each server period for the periodic and deferrable server models, with the same parameters (B, P). Theorem 1 below can be adopted to any arbitrary arrival pattern and any service time distribution (usually noted as G/G/ arrival in Kendall's notation, where "G" stands for "General") in both continuous and discrete time domains. We denote the general deferrable and periodic server queueing systems as G/G(DS)/1 and G/G(PS)/1, respectively. Without losing generality, suppose the continuous virtual waiting time processes of the deferrable and periodic queueing systems are $\tilde{V}_{DS}(t)$ and $\tilde{V}_{PS}(t)$, respectively. The virtual waiting time sequences at each start of a server period are Markov chains, $V_{DS}[n]$ and $V_{PS}[n]$, where

$$V_{DS}[n] = \tilde{V}_{DS}(nP), \quad V_{PS}[n] = \tilde{V}_{PS}(nP), \quad n \in \mathbb{N}.$$

We denote a realization of the stochastic sequences $V_{DS}[n]$ and $V_{PS}[n]$ as $v_{DS}[n]$ and $v_{PS}[n]$, respectively.

Theorem 1. *Let the same G/G/ process arrive in both the deferrable and periodic queueing servers with the same server parameters (B, P). The virtual waiting time realizations of the two systems at the start of each period are always equal to each other, i.e.,*

$$v_{DS}[n] = v_{PS}[n].$$

Fig. 4 illustrates an example of Theorem 1: We randomly generate an arriving job sequence $\{(a_i, c_i)\}$. The i-th job releases at time a_i, with execution time c_i. We use the same process to stimulate both the deferrable server and the periodic server, whose server parameters are the same $(B = 2, P = 5)$. Then, $v_{DS}(t)$ is one possible realization of the stochastic process $V_{DS}(t)$, and $v_{PS}(t)$ is a realization of $V_{PS}(t)$. By observing the virtual waiting time at the start of each period, that is, $t = 0, 5, 10, 15, 20, ...$), we find the sequence $v_{DS}[n] = v_{PS}[n]$, as Theorem 1 indicates.

To prove Theorem 1, we consider a certain server period, denoted as $[0, P)$, in Fig. 5. Suppose that we have x time units of work pending at the start of the period (time 0), and that exactly n jobs arrive within this period. The jobs are indexed in reverse: Their arrival times are $a_n, a_{n-1}, ..., a_1$, where $0 \leq a_n < a_{n-1} < ... < a_1 < P$. The execution times of the n jobs are $c_n, c_{n-1}, ..., c_1$, respectively.

We analyse this period with both the deferrable and periodic policies. We denote the work remaining at the end of this period (time P) as y_{DS} and y_{PS}, for deferrable server and periodic server, respectively.

Hypothesis H(n). In the described period, $\forall x \geq 0, \forall P > 0, \forall B \in [0, P]$, if exactly n jobs arrive within the period, then $y_{DS} = y_{PS}$. We can prove that $\forall n \in \mathbb{N}$, $H(n)$ is true.

Lemma 2. *H(0) is true.*

Proof of Lemma 2. $H(0)$ means no new job arrives within this period. We can consider only the x units of pending work.

If $x \geq B$, then in this period all of the budget will be consumed, whether DS or PS is used: i.e., $y_{DS} = y_{PS} =$

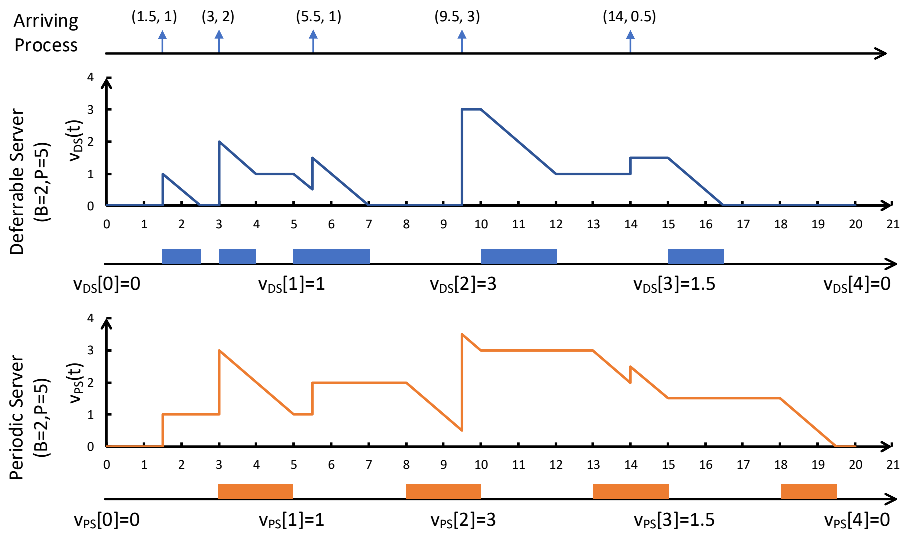


Fig. 4: Example: How $v_{DS}[n] = v_{PS}[n]$ in a possible realization

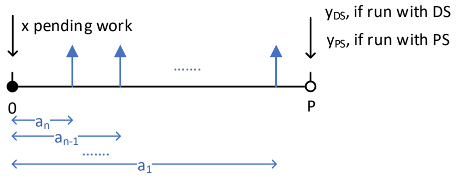


Fig. 5: n jobs fall in a server period

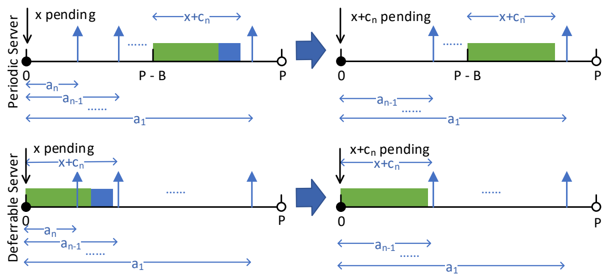


Fig. 6: Case 2.1.2: reduce the system to $n-1$ arrivals

$x - B$. If $x < B$, then $y_{DS} = y_{PS} = 0$. For either case, $y_{DS} = y_{PS} = max\{0, x - B\}$, so $H(0)$ is true. □

Lemma 3. *$H(n)$ is either true, or has the same logic value (a.k.a. truth value) as $H(n-1)$.*

Proof of Lemma 3. Considering the first job, a_n, we can enumerate the possible x and a_n values and evaluate each case. We will see that the system can be reduced to a system with $n-1$ job arrivals within the new period.

Case 1. If $x \geq B$, then $H(n)$ is true.

Whether the server is deferrable or periodic, it will exhaust exactly B time units of budget for the pending job, and all incoming jobs will accumulate. As a result,

$$y_{DS} = y_{PS} = x + \sum_{i=1}^{n} c_i - B,$$

so $H(n)$ is true.

Case 2. If $x < B$, then we must consider a_n.

Case 2.1 If $x < B$ and $a_n < x$, then we evaluate c_n.

Case 2.1.1 If $x < B$, $a_n < x$, and $x + c_n \geq B$, then $H(n)$ is true. In this case, whether the server is deferrable or periodic, it will exhaust all of its budget for x and job a_n. Effectively,

$$y_{DS} = y_{PS} = x + c_n + \sum_{i=1}^{n-1} c_i - B = x + \sum_{i=1}^{n} c_i - B,$$

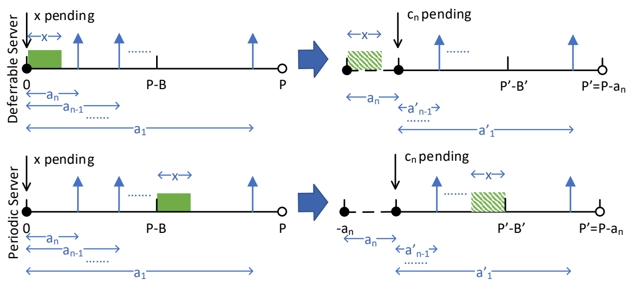
so $H(n)$ is true.

Case 2.1.2 If $x < B$, $a_n < x$, and $x + c_n < B$, then we first consider the deferrable server, as shown in Fig. 6. If $a_n < x$, then the schedule of the original system (with n arrivals) is equivalent to a system starting with a pending job of $x + c_n$, and with only $n-1$ arriving jobs. Those two systems will yield the same busy period for the deferrable server, so the pending work at the end of the period, y_{DS}, will remain unchanged.

Similarly, we can also reduce the periodic server system to a system with $n-1$ arrivals. Since the busy period stays the same, y_{PS} will remain unchanged. According to Case 2.1.1 and Case 2.1.2, for Case 2.1, $H(n)$ will have the same logic value as $H(n-1)$.

Case 2.2 If $x < B$ and $a_n \geq x$, then we consider the deferrable server first. In this case, since a_n will arrive after the pending job queue is emptied, we can reduce the system to a new system with $n-1$ arrivals with different parameters: $P' = P - a_n$, $B' = min\{B - x, P - a_n\}$, $a'_i = a_i - a_n$, and $x' = c_n$. Note that $P - a_n$ may be less than $B - x$, which means that a_n arrives so late that at least some budget has been wasted and can never be reclaimed.

Next, we consider the periodic server: As before, since a_n will arrive after the pending job queue is emptied, we can


(a) $B - x \leq P - a_n$

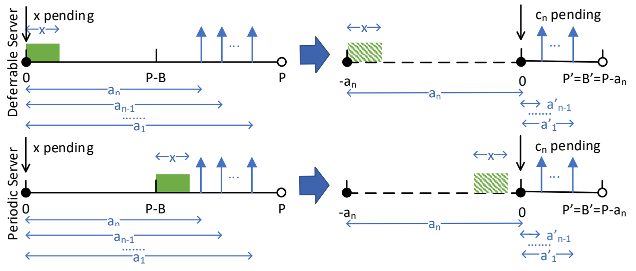

(b) $B - x > P - a_n$

Fig. 7: Case 2.2: reduce the system to $n-1$ arrivals

reduce the budget of the new system by x units and consider the new system that starts when a_n arrives. The original schedule of x must fall within the off-period of the new system. Hence, we reduce the original system to a new system with $n-1$ arrivals with different parameters: $P' = P - a_n$, $B' = min\{B - x, P - a_n\}$, $a'_i = a_i - a_n$, and $x' = c_n$.

As shown in Fig. 7, the new systems for both the deferrable server and the periodic server still satisfy the condition for $H(n-1)$: $P' \geq B$, $x' = c_n \geq 0$, and $0 \leq a_{n-1} < a_{n-1} < ... < a_1 < P$. Thus for Case 2.2, $H(n)$ has the same logic value as $H(n-1)$.

Considering Case 2.1 and Case 2.2 as a whole, for Case 2, $H(n)$ is either true or has the same logic value as $H(n-1)$.

Considering Case 1 and Case 2, $H(n)$ has the same logic value as $H(n-1)$. □

Lemma 4. *$\forall n \in \mathbb{N}$, H(n) is true.*

Proof of Lemma 4. For $n = 0$, we have proved Lemma 2. For $n > 0$, we can recursively use Lemma 3 to reduce $H(n)$ to $H(0)$, and finally to find $H(n)$ is true. □

Proof of Theorem 1. Let the system start at $t = 0$, with no initial pending job, i.e., $v_{DS}[0] = v_{PS}[0] = 0$. Using Lemma 4 recursively for each period $[kP, (k+1)P)$, we can conclude that $v_{DS}[n] = v_{PS}[n]$. □

The stationary virtual waiting times of the two Markov chains are $V_{DS}[n] \xrightarrow{p} V_{DS}$, and $V_{PS}[n] \xrightarrow{p} V_{PS}$, respectively.

Corollary 5. *Let a G/G/ process arrive in both the deferrable and periodic servers with the same server parameters (B, P). If the stability condition holds, the stationary distributions of the virtual waiting times of the two systems at the start of each period exhibit* ***almost sure equality***[4]*, i.e.:*

$$V_{DS} \overset{\text{a.s.}}{=} V_{PS}.$$

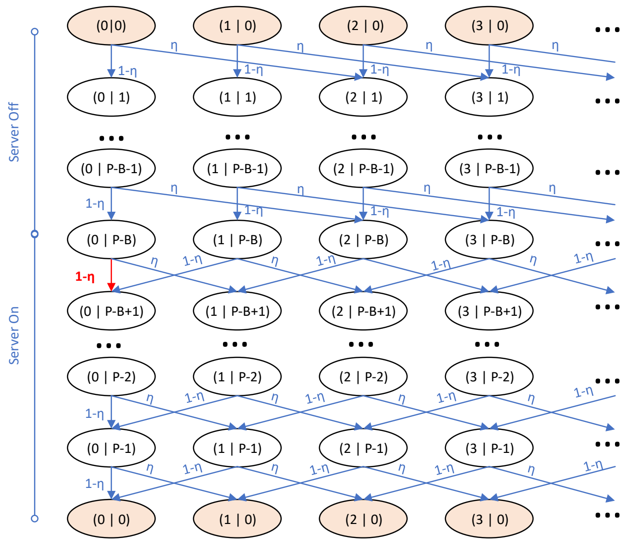


Fig. 8: State Transition Diagram for a B/D(PS)/1 queue: $Pr\{\overline{V}_{PS} = l | T = n\}$, $d = 2$

Proof of Corollary 5. Using Theorem 1, we have $v_{DS}[n] = v_{PS}[n]$. Thus, $Pr\{V_{DS}[n] = V_{PS}[n]\} = 1$. That is, $V_{DS}[n] \overset{\text{a.s.}}{=} V_{PS}[n]$. If the stability condition holds, statistical equilibrium can be achieved: $V_{DS} \overset{\text{a.s.}}{=} V_{PS}$. □

C. Conditional Stationary Virtual Waiting Time in a Period

Corollary 5 indicates that a comparable deferrable server and periodic server have the same stationary virtual waiting times at the start of a server period: $\overline{V}_{DS}|(T = 0)$ has the same distribution as $\overline{V}_{PS}|(T = 0)$. We apply Corollary 5 on B/D/ arrival to simplify the computation of the stationary response time distribution of a B/D(DS)/1 queue.

Stationary Virtual Waiting Time of a Periodic Server. Supposing the B/D(PS)/1 queue reaches its statistical equilibrium, we denote the distribution of stationary virtual waiting times at any slot of the server period as $p_{l,n}$, where

$$p_{l,n} = Pr\{\overline{V}_{PS} = l | T = n\} \quad l, n \in \mathbb{N}. \tag{2}$$

Then, when the sever is in the off-state $(0 \leq n \leq P-B-1)$,

$$\begin{cases} p_{l,n+1} = (1-\eta)p_{l,n} & 0 \leq l < d, \\ p_{l,n+1} = (1-\eta)p_{l,n} + \eta p_{l-d,n} & l \geq d. \end{cases} \tag{3}$$

When the sever is in the on-state $(P - B \leq n \leq P)$,

$$\begin{cases} p_{0,n+1} = (1-\eta)(p_{0,n} + p_{1,n}) & l = 0, \\ p_{l,n+1} = (1-\eta)p_{l+1,n} & 1 \leq l < d-1, \\ p_{l,n+1} = (1-\eta)p_{l+1,n} + \eta p_{l-d+1,n} & l \leq d-1. \end{cases} \tag{4}$$

[4]In probability and statistics, *almost sure equality* is a stronger equivalence than *equality in distribution* [40].

Equations (3) and (4) govern the state transitions within a server period. Fig. 8 shows an example of the state transition within a server period of the B/D(PS)/1 queue when $d = 0$.

If the statistical equilibrium has been achieved, due to the periodicity of the system, we have

$$p_{l,n+P} = p_{l,n}, \quad \forall l \in \mathbb{N}. \tag{5}$$

Consider the normalization condition of the conditional probability mass function:

$$\sum_{l=0}^{\infty} p_{l,n} = \sum_{l=0}^{\infty} Pr\{\tilde{V}_{PS} = l | T = n\} = 1, \quad 0 \le n \le P. \tag{6}$$

Equations (3), (4), (5), and (6) form a differential equation system, and $p_{l,n}$ can be uniquely determined. However, instead of deriving a purely analytical solution for the differential system, we employ a positive convergent series, $\{p_{l,n}\}_{l=0}^{\infty}$, and use the mathematical nature of this series [42] to construct a practical algorithm.

Theorem 6. $\forall \epsilon > 0, n \in \mathbb{N}, \exists M \in \mathbb{N}, s.t. \sum_{l=M}^{\infty} p_{l,n} < \epsilon$.

Proof of Theorem 6. According to equation (6), the non-negative series $\{p_{l,n}\}$ is convergent. The partial summation series $\{S_m\}_{m=0}^{\infty} \to 1$, where $S_m = \sum_{l=0}^{m} p_{l,n}$. In other words, $\forall \epsilon > 0, n \in \mathbb{N}, \exists M \in \mathbb{N}, s.t.$

$$\sum_{l=0}^{M} p_{l,n} > 1 - \epsilon. \tag{7}$$

Using (6) minus (7), we get $\sum_{l=M}^{\infty} p_{l,n} < \epsilon$. □

Theorem 6 indicates that in practice, if we choose a large enough M, we can ignore the tail distribution of $p_{l,n}$. This feature helps us construct a practical algorithm: We focus on only the first M items, i.e., $\{p_{l,n}\}_{l=0}^{M-1}$, and treat the tails, $\{p_{l,n}\}_{l=M}^{\infty}$, as 0s.

Algorithm 1 demonstrates how we leverage the recursive structure in Fig. 8 to compute $\{p_{l,0}\}_{l=0}^{M-1}$: (i) we initialize only $p_{0,0} = 1$, while others are equal to 0, which represents layer $n = 0$ of Fig. 8; (ii) using equation (3), we can reach layer $n = P - B$. Using equation (4), we can reach layer $n = P$; (iii) because of equation (5), layer P is effectively layer 0. By normalizing the result of layer P, we are able to compare it to the original layer 0; (iv) we repeat the procedure until the 1-norm error is less than the threshold ($\delta < \Delta$).

Algorithm 1 also indicates how to compute $\{p_{l,n}\}_{l=0}^{M-1}, n = 1, 2, ..., P-1$: After $\{p_{l,0}\}_{l=0}^{M-1}$ converges, we can use equations (3) and (4) to compute and then normalize each layer.

Stationary Virtual Waiting Time of a Deferrable Server. In contrast to a periodic server, the active time slot of a deferrable server is no longer independent of the arrival process, and the remaining budget must be included in the state transition diagram. Thus, we express the conditional joint distribution of both the virtual waiting time $\tilde{V}_{DS}$ and the remaining budget G,

$$q_{l,g,n} = Pr\{\tilde{V}_{DS} = l, G = g | T = n\}, \quad l, n \in \mathbb{N}, 0 \le g \le B. \tag{8}$$

Algorithm 1: Compute first M terms of $p_{l,0}$

```
   Input: Vector Length: $M$, Expected Error: $\Delta$
   Output: $M \times$ 1-Vector: $\overrightarrow{V} = (p_{0,0}, p_{1,0}, ..., p_{M-1,0})$
1  $M \times$ 1-Vector: $\overrightarrow{V} \leftarrow (1, 0, 0, ...)$;   $\delta \leftarrow \infty$;
2  while $\delta \ge \Delta$ do
3      $\overrightarrow{V'} \leftarrow \overrightarrow{V}$;
4      for $i \leftarrow 0$ to $P - B - 1$ do
5          $M \times$ 1-Vector: $\overrightarrow{V_T} \leftarrow (0, 0, 0, ...)$;
6          Compute $\overrightarrow{V_T} = (p_{0,i+1}, p_{1,i+1}, ..., p_{M-1,i+1})$
             from $\overrightarrow{V} = (p_{0,i}, p_{1,i}, ..., p_{M-1,i})$, using Eq.(3);
7          $\overrightarrow{V} \leftarrow \overrightarrow{V_T}$;
8      end
9      for $i \leftarrow P - B$ to $P - 1$ do
10         $M \times$ 1-Vector: $\overrightarrow{V_T} \leftarrow (0, 0, 0, ...)$;
11         Compute $\overrightarrow{V_T} = (p_{0,i+1}, p_{1,i+1}, ..., p_{M-1,i+1})$
             from $\overrightarrow{V} = (p_{0,i}, p_{1,i}, ..., p_{M-1,i})$, using Eq.(4);
12         $\overrightarrow{V} \leftarrow \overrightarrow{V_T}$;
13     end
14     $\overrightarrow{V} \leftarrow \overrightarrow{V} / \|\overrightarrow{V}\|_1$;   $\delta \leftarrow \|\overrightarrow{V} - \overrightarrow{V'}\|_1$;
15 end
16 return $\overrightarrow{V}$;
```

A deferrable server replenishes its budget at the start of each period, which means the remaining budget must be B when $n = 0$. According to Theorem 1, we have

$$\begin{cases} q_{l,B,0} = p_{l,0}, & l \in \mathbb{N}, \\ q_{l,g,0} = 0, & l \in \mathbb{N}, g < B. \end{cases} \tag{9}$$

Now we compute the conditional joint distribution $q_{l,g,n}$ for layers $n \ge 1$. Because of the deferrable server policy and additional state variable G, the B/D(DS)/1 queue has B times the number of states as in B/D(PS)/1. Fortunately, many of those states are not reachable. Thus, instead of deriving equations corresponding to (3) and (4) for a B/D(DS)/1 system, we can use a forward recursion algorithm (Algorithm 2), which helps us determine both the effective states and the probability.

The algorithm returns a $M \times B \times P$-Matrix, A, each item of which represents $q_{l,g,n}$, and a list, `nz_list`, comprised of n sets, with each set recording the unique non-zero states of layer n.

Algorithm 2 iterates through the layers, and in each layer n, we iterate through each non-zero state. For each state, we have probability η of an incoming job and $1 - \eta$ for no incoming job. Each non-zero state can potentially transit to two different states in the next layer. We determine the state for the next layer by evaluating state parameters l and g with the deferrable policy, and finally we get the matrix and the `nz_list`.

How Theorem 1 Helps Reduce Complexity. One can derive a state transition diagram for B/D(DS)/1 and use a similar iteration method as in Algorithm 1 to compute the stationary distribution. However, such a method needs to iterate through each state of the deferrable server, which is $O(MP^2)$. If

Algorithm 2: Compute $q_{l,g,n}$

```
Input: $M \times 1$-Vector: $\overrightarrow{V} = (p_{0,0}, p_{1,0}, ..., p_{M-1,0})$
Output: $M \times B \times P$-Matrix: $A = \{q_{l,g,n}\}$, List: nz_list
A = zeros($M, B, P$); nz_list ← [ ]; tset ← { };
for $l \leftarrow 0$ to $M-1$ do
    $A[l][B][0] \leftarrow p_{l,0}$;
    tset.union($\{(l, B)\}$);
end
nz_list.append(tset);
for $n \leftarrow 0$ to $P-2$ do
    iterset ← nz_list[$n$]; tset ← { };
    for $(l, g)$ in iterset do
        if $g > 0$ then
            if $l > 0$ then
                $A[l-1][g-1][n+1]$ += $(1-\eta)A[l][g][n]$;
                tset.union($\{(l-1, g-1)\}$);
                if $l + d - 1 \leq M$ then
                    $A[l+d-1][g-1][n+1]$ += $\eta A[l][g][n]$;
                    tset.union($\{(l+d-1, g-1)\}$);
                end
            else
                $A[l][g][n+1]$ += $(1-\eta)A[l][g][n]$;
                tset.union($\{(l, g)\}$);
                if $l + d - 1 \leq M$ then
                    $A[l+d-1][g-1][n+1]$ += $\eta A[l][g][n]$;
                    tset.union($\{(l+d-1, g-1)\}$);
                end
            end
        else
            $A[l][g][n+1]$ += $(1-\eta)A[l][g][n]$;
            tset.union($\{(l, g)\}$);
            if $l + d > M$ then
                $A[l+d][g][n+1]$ += $\eta A[l][g][n]$;
                tset.union($\{(l+d, g)\}$);
            end
        end
    end
    nz_list.append(tset);
end
return $(A, \text{nz\_list})$;
```

the outer "while" loop needs K iterations, then the overall complexity is $O(KMP^2)$. Thanks to Theorem 1, we can first compute $\{p_{l,n}\}_{l=0}^{M-1}$ by Algorithm 1 with $O(K \times M \times P)$, then explore the state space of B/D(DS)/1 only once by Algorithm 2 with $O(MP^2)$. Hence we get an overall complexity of $O(MP(K+P))$ rather than $O(KMP^2)$.

D. Determine the Stationary Response Time Distribution

Finally, we consider the stationary response time distribution, $F_{DS}(t) = Pr\{R_{DS} = t\}$, for a B/D(DS)/1 queue.

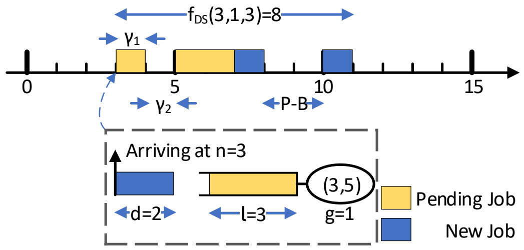


Fig. 9: Determine the Response Time, An Example

Using the total probability law over the joint random variables $\overline{V}_{DS}, G, T$:

$$F_{DS}(t) = Pr\{R_{DS} = t\} = \sum_{l=0}^{\infty}\sum_{g=0}^{B}\sum_{n=0}^{P-1} Pr\{R_{DS} = t | \overline{V}_{DS} = l, G = g, T = n\} Pr\{\overline{V}_{DS} = l, G = g, T = n\}. \tag{10}$$

Given the values of (l, g, n), the response time can be determined directly by $f_{DS}(l, g, n)$, where

$$f_{DS}(l,g,n) = \begin{cases} h, & h \leq \gamma_1, \\ h + \gamma_2, & \gamma_1 < h \leq \gamma_1 + B, \\ h + \gamma_2 + \lceil \frac{h - (\gamma_1 + B)}{B} \rceil (P - B), & h > \gamma_1 + B. \end{cases} \tag{11}$$
$$h = l + d, \quad \gamma_1 = \min\{P - n, g\}, \quad \gamma_2 = P - n - \gamma_1.$$

We use an example to illustrate both the notation and the intuition of equation (11). Fig. 9 shows a B/D(DS)/1 system with $P = 5$, $B = 3$, and $d = 2$. At the third slot of a period ($n = 3$), a new incoming job arrives, and the sever has only one unit of budget remaining ($g = 1$) for current period, while it still has three-unit pending jobs ($l = 3$). We want to figure out the response time of the new incoming job, i.e., $f_{DS}(3, 1, 3)$.

The server needs to provide h units of service to finish the new incoming job. γ_1 indicates the available service units within the first period. Clearly, if the server can provide enough service time within the first period ($h \leq \gamma_1$), the response time should be h, as in the first case of equation (11).

Then, if the h units can be fulfilled before the end of the second period ($h < \gamma_1 + B$), we need to pay only an additional γ_2 budget replenishment penalty. Thus the response time is $h + \gamma_2$, as in the second case of equation (11).

If more periods are required to fulfill the h units of service time, we need to wait $P-B$ units each time, before consuming B units service from each new period.

In Fig. 9, the service time cannot be fulfilled within the first two periods ($h = 5 > \gamma_1 + B = 4$). We need one ($\lceil \frac{5-(1+3)}{3} \rceil = 1$) more period besides the first two, which we get by waiting for one additional budget replenishment ($P - B = 2$) . As a result, the response time is $5 + 1 + 1 \times 2 = 8$.

Since $f_{DS}(l, g, n)$ is determined, the first factor in equation (10), $Pr\{R_{DS} = t | \overline{V}_{DS} = l, G = g, T = n\}$, is either 1 or 0. The first factor equals 1, only if $f_{DS}(l, g, n) = t$.

To obtain the second factor in equation (10), using Bayes' theorem, the PASTA theorem [31], and equation (8), we have

$$Pr\{\overline{V}_{DS} = l, G = g, T = n\} = \frac{q_{l,g,n}}{P}. \quad (12)$$

Finally, the response time distribution of B/D(DS)/1 can be represented as

$$F_{DS}(t) = \frac{1}{P} \sum_{\substack{l \geq 0, 0 \leq g \leq B, 0 \leq n \leq P-1, \\ f_{DS}(l,g,n)=t}} q_{l,g,n}. \quad (13)$$

E. Summary of the Numerical Method

We now summarize how to compute the stationary response time distribution of an M/D(DS)/1 queue with the parameters (λ, d', B', P'), as follows:

Given an M/D(DS)/1 system: (λ, d', B', P'),
1. Choose N, using Table I for a B/D(DS)/1 system (η, d, B, P);
2. Use Algorithm 1 to compute the stationary virtual waiting time distribution at the start of a server period ($q_{l,B,0}$ of Equation (9));
3. Use Algorithm 2 to compute the conditional stationary virtual waiting time distribution at each slot within a period ($\{q_{l,g,n}\}$) recursively;
4. Compute the stationary response time distribution $F_{DS}(t)$ via Equation (13). □

V. Configuration to meet SLOs

In this section, we apply our algorithm as a design tool, showing how the computed stationary response time distribution can be used to meet the service level objective (SLO) of a time-sensitive service.

Latency SLO Validation. A time-sensitive service may have a required tail latency performance: e.g., a 90^{th} percentile latency of less than $10ms$. This latency objective can be represented as a point $(d_0 = 10ms, p = 0.90)$ on the same coordinate system as a stationary response distribution whose x-axis is latency and y-axis is cumulative proportion. For an M/D(DS)/1 system with the parameters (λ, d, B, P), we can uniquely determine the CDF of its stationary response time. This capability makes SLO validation trivial: if the point (d_0, p) falls below the CDF curve, then the SLO requirement will be met; otherwise, the SLO cannot be met. The single-point latency objective can also be generalized to a multi-point latency objective and even to a lower-bound CDF curve objective.

Parameter Space Exploration. Computing the stationary response time distribution enables us to conduct both qualitative and quantitative analysis for an M/D(DS)/1 queue. As shown in Fig. 10, to illustrate the impact of each parameter, we change only one parameter while keeping the others unchanged. We plot the M/D/1 response time distribution [38] as a reference when sweeping W and P. We make three observations: (1) When λ and d increase, the utilization $U = \lambda d$ increases. As a result, the response time performance becomes worse (Fig. 10a and 10b). (2) As shown in Fig. 10c, when λ, d, and P are constant, the larger the budget (bandwidth), the better the response time. When $B = P$ and hence $W = 100\%$, the M/D(DS)/1 model degenerates to an M/D/1 model. Indeed, since the server of the M/D(DS)/1 queueing model is subject to a resource constrained policy (deferrable server with the parameters (B, P)), we can immediately conclude that the response time distribution of M/D(DS)/1 is upper bounded by that of the M/D/1 queue. (3) As shown in Fig. 10d, when λ, d and $W = \frac{B}{P}$ are constant, the larger the period (and the budget, proportionally), the more the deferrable server can tolerate the burstiness introduced by stochastic arrivals. As a result, we get better response time performance. If $P \to \infty$, the M/D(DS)/1 system degenerates to an M/D/1 model.

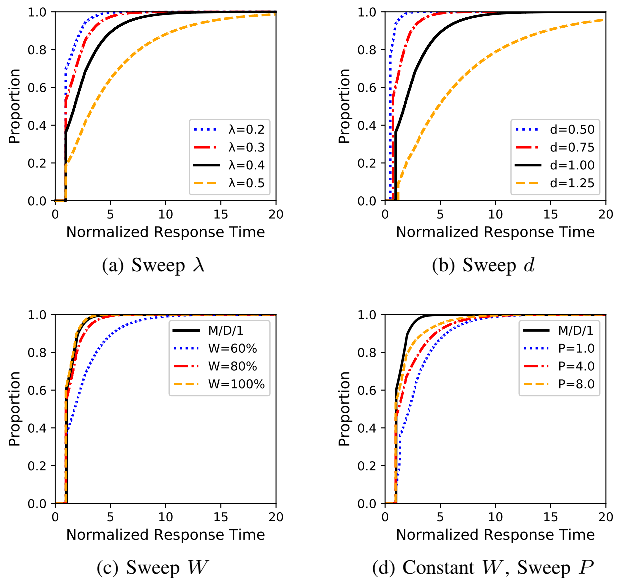
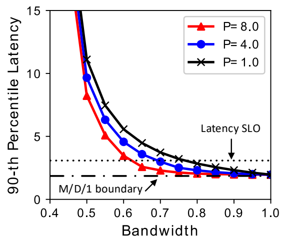


Fig. 10: Response Time Distribution and Parameter Sweep, Base Condition $\lambda = 0.4, d = 1.0, P = 2.0, B = 1.2$

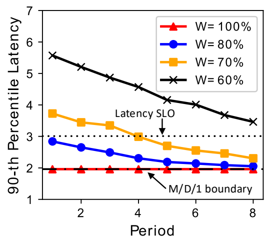
Parameter Selection. In practice, the job arrival rate (λ) and service duration (d) usually cannot be easily adjusted, while the server period (P) and budget (B) can be configured to meet a tail latency requirement, noted as (d_0, p). With the M/D(DS)/1 stationary response time distribution, parameter selection is straightforward, as follows:

(1) SLO feasibility: The response time distribution of the M/D/1 provide an upper bound of the M/D(DS)/1 response time distribution. Only if $B = P$ can the M/D(DS)/1 achieve the same performance of the M/D/1 queue. If the SLO (d_0, p) falls below the M/D/1 response time distribution, we may be able to manipulate (B, P) of an M/D(DS)/1 queue to achieve the goal. Otherwise, the SLO is unachievable.

(2) Selecting (B, P): Suppose the SLO is achievable, and we want to provide a 90^{th} percentile latency of less than 3, i.e., a SLO $(d_0 = 3, p = 0.90)$ with an arrival rate of



(a) Choosing W



(b) Choosing P

Fig. 11: Parameter Selection, with $\lambda = 0.4, d = 1.0$

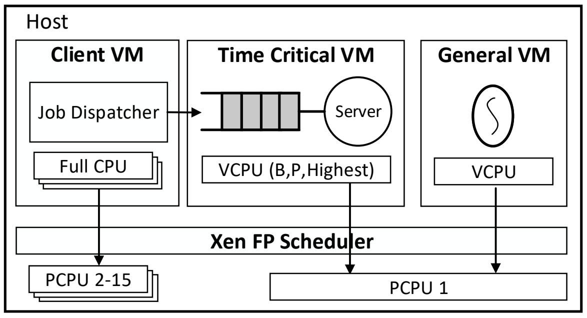


Fig. 12: M/D(DS)/1 System for Evaluation

$\lambda = 0.4$ and a constant service duration $d = 1.0$. We can directly plot the curves as in Fig. 11 and focus on how the 90^{th} percentile latency changes versus W and P. By choosing a proper bandwidth value W and period value P (so that $B = PW$ is effectively determined), the desired latency can be made less than the goal $d_0 = 3$ defined in the SLO.

Specifically, if overprovisioning by increasing the server bandwidth is acceptable, a drastic decrease in the tail latency can be achieved (Fig. 11a). For example, choosing $P = 4.0$, any $W \geq 0.7$ can satisfy the latency SLO. Latency also can be improved by keeping the same bandwidth and increasing the period and budget proportionally (Fig. 11b). For example, by choosing $W = 0.7$, any $P \geq 4.0$ can satisfy the latency SLO.

Again, none of these methods can yield a better latency than the M/D/1 limit: In Fig. 11a, all the curves (with different P) cross when $W = 100\%$ (i.e., $P = B$), and hence the system degenerates to an M/D/1 system and reaches the M/D/1 boundary. Similarly, in Fig. 11b, the curve $W = 100\%$ overlaps with the M/D/1 boundary, and other curves converge to the M/D/1 boundary when $P \to \infty$. However, a large P implies that a time-sensitive service may potentially monopolize the CPU for a long time. As a result, it will have negative impacts on general services which share the same CPU. Our analysis allows operators to select the period and budget to meet the tail latency without a unnecessarily large period.

VI. Evaluation

In this section, we evaluate our algorithm as a predictive tool. By comparing our numerical results with empirical results on real systems, we can assess how closely the numerical prediction can approximate the empirical results.

We conducted experiments on a machine with one Intel E5-2683v4 16-core CPU and 64 GB memory. We disabled hyper-threading and power saving features and fixed the CPU frequency at 2.1 GHz to reduce unpredictability, as in [7], [43], [44]. We ran time-critical services and general services in Linux virtual machines (VMs) on the Xen 4.10.0 hypervisor. We modified the *Real-Time Deferrable Server* (RTDS) to support a partitioned fixed priority scheduling policy[5] [7], [8]. The modified RTDS scheduler treated each VCPU as a deferrable server with three parameters: budget, period, and priority. We used Linux 4.4.19 for all VMs. We configured Domain 0 with one full CPU pinned to one dedicated core, i.e., PCPU 0.

[5]The vanilla RTDS is a gEDF scheduler.

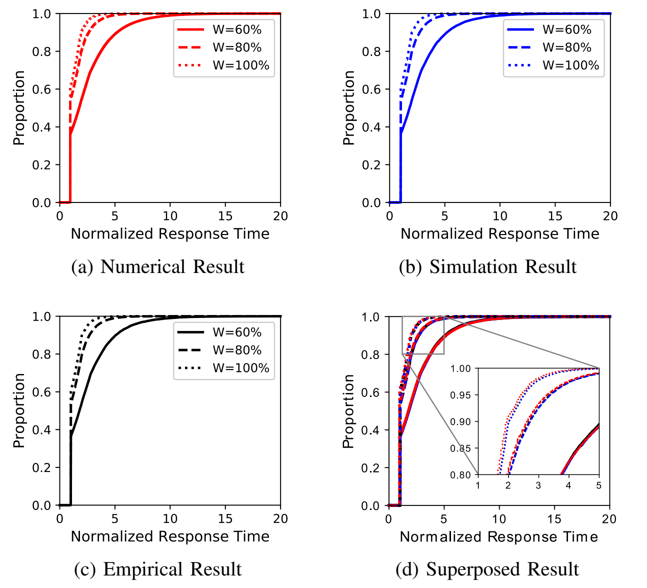
As shown in Fig. 12, the time-critical service ran on a single VCPU virtual machine, which had highest priority. The VCPU shared the same PCPU1 with another VCPU which was running a "CPU-Hog" process in a general VM. We used another 14-VCPU VM to run clients that dispatched jobs to the service under test.

We used two different services for the experiments: a synthetic server and a Redis [3] server. The synthetic server is designed to have a predictable execution time with litter variance. The Redis server worked as a representative time-critical service commonly used in cloud environments.

Synthetic Server. We used four parameters, (λ, d, B, P), to represent an M/D(DS)/1 system. For each configuration of the system, we measured the response time distribution in three different ways: (1) We directly computed the response time distribution via our algorithm. (2) We randomly generated $20,000$ samples following a Poisson process with a rate of λ, and then simulated the system behavior to get the response time distribution. (3) Using the same Poisson process, we let the job dispatcher stimulate the time-critical service on the testbed as shown in Fig. 12, then measured the empirical response time distribution. We expected that those three approaches would produce similar results.

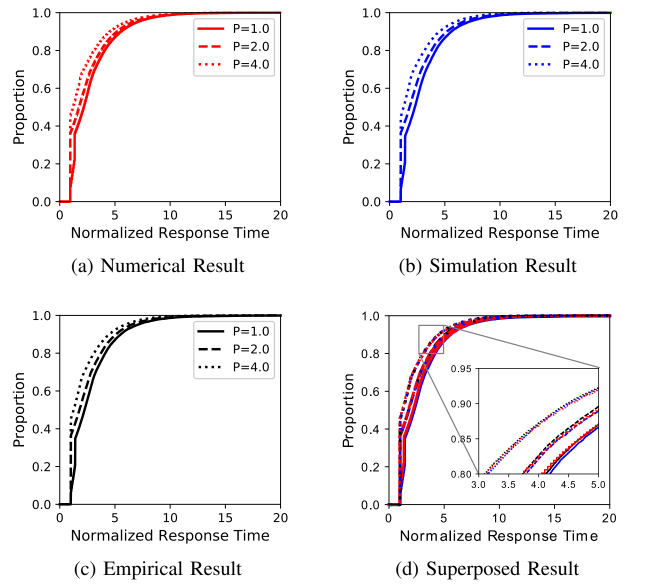
We let the arrival rate be $\lambda = 0.004\ event/ms$, and the constant service duration be $d = 100ms$. We first fixed the period $P = 200ms$, while we varied B over $\{120ms, 160ms, 200ms\}$, corresponding to a bandwidth W of $\{60\%, 80\%, 100\%\}$. We chose $N = 20$ for our algorithm.

Fig. 13a, 13b, and 13c show the numerical, simulation, and empirical results. Fig. 13d is the superposition of the first three figures, and it supports two observations. (1) The numerical results, simulation results, and empirical results closely approximate each other. The fact that our numerical results approximate the simulated ones validates the correctness of our numerical algorithm. Both results also approximate the empirical one, indicating that our system and synthetic server implementation fit the M/D(DS)/1 model. (2) For the same period value, the larger the bandwidth, the better the response



(a) Numerical Result (b) Simulation Result
(c) Empirical Result (d) Superposed Result

Fig. 13: Response Time Distribution of Synthetic Server, Fixed $P = 200ms$, Results Normalized against $d = 100ms$



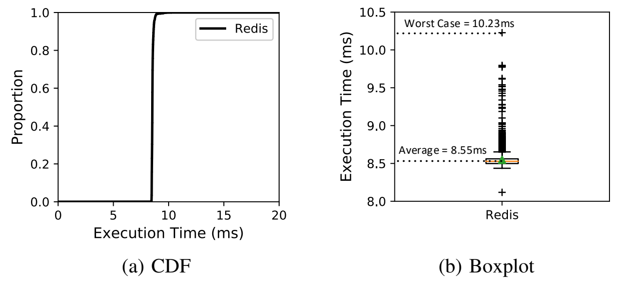
(a) Numerical Result (b) Simulation Result
(c) Empirical Result (d) Superposed Result

Fig. 14: Response Time Distribution of Synthetic Server, Fixed $W = 60\%$, Results Normalized against $d = 100ms$

time distribution.

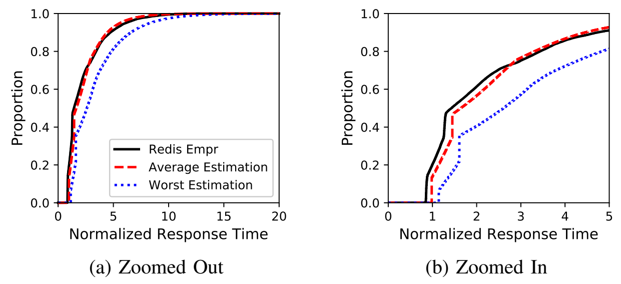
Keeping the same arrival rate and service duration, we then fixed the bandwidth $W = 60\%$, while varying P (and B proportionally) over $\{100ms, 200ms, 400ms\}$.

As shown in Fig. 14, these results support two observations: (1) The numerical, simulation, and empirical results closely approximate each other. (2) Given the same bandwidth value, the larger the period, the better the response time distribution.



(a) CDF (b) Boxplot

Fig. 15: Redis Execution Time Distribution



(a) Zoomed Out (b) Zoomed In

Fig. 16: Response Time Distribution of Redis, $P = 10ms$, $B = 6ms$, Normalized against $d = 8.55ms$

Redis. Being a single-threaded in-memory data storage server, Redis is typically deployed as a micro-service or in a virtualized host, as in AWS ElastiCache for Redis [45]. We used Redis as a time-sensitive workload to verify whether our approach can effectively predict the stationary response time distribution for a real-world application.

We used the Redis server in a time-sensitive VM as shown in Fig. 12. We chose the "SORT" query, which sorts a 20,000-item key-value based list. Unlike the synthetic server, whose job execution times can be calibrated and fine-tuned, we measured the execution time for Redis empirically. Fig. 15 shows the execution time distribution of the Redis "SORT" query. Though the service time is relatively predictable (Fig. 15a), but with non-negligible variance (Fig. 15b): The average execution time and worst case execution time were $8.55ms$ and $10.23ms$, respectively. We used the average time for predicting an approximated distribution, and the worst case value for predicting the lower bound of the response time distribution.

We stimulated the system with a job arrival rate of $\lambda = 0.004\ event/ms$, setting the deferrable server period $P = 10ms$ and $B = 6ms$. We estimated the response time distribution by using an M/D(DS)/1 model with our numerical analysis. First, using the average execution time for Redis ($d = 8.55ms$) as an approximation of the deterministic service duration in an M/D(DS)/1 system, we computed the response distribution by using our numerical approach. Second, using the worst case execution time ($d = 10.23ms$), we derived a lower bound of the response time distribution.

As shown in Fig. 16, we observed that the response time distribution, when using the average Redis execution time ($d = 8.55ms$) as the deterministic execution duration, approximated the empirical result closely. However, due to the variability of the Redis execution time distribution (Fig. 15b), the head of the empirical distribution is better than estimated, while the tail portion is worse. In comparison, when using the worst case execution time for estimation, the estimated response time distribution provided a lower bound of the empirical distribution.

VII. Related Work

Kaczynski, Lo Bello, and Nolte [21] extended the SAF model [14] by allowing aperiodic tasks to run within polling servers. The objectives of our work and their work are different. On one hand, the extended SAF model was not designed for deriving the exact response time distribution of aperiodic tasks, which is the main objective of our work. On the other hand, another aspect of the extended SAF model is more general: It allows arbitrary arriving and arbitrary execution for aperiodic task, by using *Arrival Profile* and *Execution Time Profile (ETP)*, and running them within a polling server with any priority, while our system allows the aperiodic task to be a Poisson arrival with deterministic execution, running within a highest priority deferrable server. Unfortunately, given the difference between the deferrable server and the polling server, the ETP extraction cannot be directly adopted on a deferrable server.

Queueing systems with periodic service have been studied since 1956 [37]. Researchers encountered mathematical difficulties when trying purely analytical techniques to study virtual waiting time and response time distributions on continuous time domain models [34]–[36]. Eenige [33] focused on discrete time queueing systems with periodic services. He observed the mathematical difficulties in deriving a pure analytical method even on a simplistic B/D(PS)/1 queue whose service time equals one time slot, i.e., $d = 1$. As a result, Eenige employed numerical methods for calculating the virtual waiting time distribution for a discrete queueing system with periodic service. Inspired by the queueing model with periodic service and virtual waiting time analysis, here we first observe the virtual waiting time equivalence between the periodic server and the deferrable server. Combining scheduling analysis and virtual waiting time analysis, we derive an efficient method to compute the stationary response time distributions of M/D(DS)/1 models.

VIII. Conclusions

The proliferation of time-critical services in cloud computing has emphasized the importance of performance isolation. In this paper we consider a time-critical service scheduled as a deferrable server for performance isolation, and we develop a numerical method for computing the stationary response time distribution of a stochastic Poisson arrival process. The numerical method takes advantage of the equivalence of virtual waiting times between periodic and deferrable server. Knowing the stationary response time distribution of the M/D(DS)/1 queue, we demonstrated how the method can enable an operator to meet the latency SLO of a time-critical service. We implemented a prototype testbed based on Xen 4.10.0 hypervisor and evaluated two case studies involving both a synthetic service and the commonly used Redis service. The results of those studies demonstrated the accuracy of our approach as a predictive tool in supporting time-sensitive services in practical real-world settings. In the future, we aim to extend the system model by allowing arbitrary job execution times with a known distribution, i.e., an M/G(DS)/1 queue .

Acknowledgments

This research was supported in part by NSF grant 1646579 (CPS) and the Fullgraf Foundation.

References

[1] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[2] J. Li, K. Agrawal, S. Elnikety, Y. He, I. Lee, C. Lu, K. S. McKinley *et al.*, "Work stealing for interactive services to meet target latency," in *ACM SIGPLAN Notices*, vol. 51, no. 8. ACM, 2016, p. 14.

[3] "Introduction to Redis," https://redis.io/topics/introduction, 2018.

[4] D. Namiot and M. Sneps-Sneppe, "On micro-services architecture," *International Journal of Open Information Technologies*, vol. 2, no. 9, pp. 24–27, 2014.

[5] Y. Li, R. West, Z. Cheng, and E. Missimer, "Predictable communication and migration in the quest-v separation kernel," in *Real-Time Systems Symposium (RTSS)*, 2014.

[6] J. Lelli, G. Lipari, D. Faggioli, and T. Cucinotta, "An efficient and scalable implementation of global edf in linux," in *proc. of the 7th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT11)*. Citeseer, 2011, pp. 6–15.

[7] S. Xi, M. Xu, C. Lu, L. T. Phan, C. Gill, O. Sokolsky, and I. Lee, "Real-time multi-core virtual machine scheduling in Xen," in *2014 International Conference on Embedded Software (EMSOFT)*, 2014.

[8] M. Xu, "RTDS-based-scheduler," https://wiki.xenproject.org/wiki/RTDS-Based-Scheduler, 2015.

[9] A. F. Mills and J. H. Anderson, "A stochastic framework for multiprocessor soft real-time scheduling," in *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2010, pp. 311–320.

[10] J. M. Calandrino, D. Baumberger, T. Li, S. Hahn, and J. H. Anderson, "Soft real-time scheduling on performance asymmetric multicore platforms," in *13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'07)*. IEEE, 2007, pp. 101–112.

[11] J. P. Erickson, G. Coombe, and J. H. Anderson, "Soft real-time scheduling in google earth," in *2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*. IEEE, 2012, pp. 141–150.

[12] B. Tanasa, U. D. Bordoloi, P. Eles, and Z. Peng, "Probabilistic response time and joint analysis of periodic tasks," in *2015 27th Euromicro Conference on Real-Time Systems*. IEEE, 2015, pp. 235–246.

[13] L. Abeni, N. Manica, and L. Palopoli, "Efficient and robust probabilistic guarantees for real-time tasks," *Journal of Systems and Software*, vol. 85, no. 5, pp. 1147–1156, 2012.

[14] J. L. Díaz, D. F. García, K. Kim, C.-G. Lee, L. L. Bello, J. M. López, S. L. Min, and O. Mirabella, "Stochastic analysis of periodic real-time systems," in *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.* IEEE, 2002, pp. 289–300.

[15] Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean, "A statistical response-time analysis of real-time embedded systems," in *2012 IEEE 33rd Real-Time Systems Symposium*. IEEE, 2012, pp. 351–362.

[16] D. Maxim and L. Cucu-Grosjean, "Response time analysis for fixed-priority tasks with multiple probabilistic parameters," in *2013 IEEE 34th Real-Time Systems Symposium*. IEEE, 2013, pp. 224–235.

[17] L. Palopoli, D. Fontanelli, N. Manica, and L. Abeni, "An analytical bound for probabilistic deadlines," in *2012 24th Euromicro Conference on Real-Time Systems*. IEEE, 2012, pp. 179–188.

[18] B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic task scheduling for hard-real-time systems," *Real-Time Systems*, vol. 1, no. 1, pp. 27–60, 1989.

[19] J. K. Strosnider, J. P. Lehoczky, and L. Sha, "The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments," *IEEE Transactions on Computers*, vol. 44, no. 1, pp. 73–91, 1995.

[20] T.-H. Lin and W. Tarng, "Scheduling periodic and aperiodic tasks in hard real-time computing systems," in *ACM SIGMETRICS performance evaluation review*, vol. 19, no. 1. ACM, 1991, pp. 31–38.

[21] G. A. Kaczynski, L. Lo Bello, and T. Nolte, "Deriving exact stochastic response times of periodic tasks in hybrid priority-driven soft real-time systems," in *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*, Sep. 2007, pp. 101–110.

[22] T. F. Abdelzaher, V. Sharma, and C. Lu, "A utilization bound for aperiodic tasks and priority driven scheduling," *IEEE Transactions on Computers*, vol. 53, no. 3, pp. 334–350, 2004.

[23] I. Shin and I. Lee, "Compositional real-time scheduling framework," in *Real-Time Systems Symposium (RTSS)*, 2004.

[24] A. Easwaran, I. Shin, and I. Lee, "Optimal virtual cluster-based multiprocessor scheduling," *Real-Time Systems*, vol. 43, no. 1, pp. 25–59, 2009.

[25] S. K. Baruah and N. Fisher, "Component-based design in multiprocessor real-time systems," in *International Conference on Embedded Software and Systems (ICESS)*, 2009.

[26] I. Shin, A. Easwaran, and I. Lee, "Hierarchical scheduling framework for virtual clustering of multiprocessors," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2008.

[27] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *2000 IEEE International Symposium on Circuits and Systems. Emerging Technologies for the 21st Century. Proceedings (IEEE Cat No. 00CH36353)*, vol. 4. IEEE, 2000, pp. 101–104.

[28] L. T. Phan, S. Chakraborty, and P. Thiagarajan, "A multi-mode real-time calculus," in *2008 Real-Time Systems Symposium*. IEEE, 2008, pp. 59–69.

[29] J. P. Lehoczky, "Real-time queueing theory," in *17th IEEE Real-Time Systems Symposium*. IEEE, 1996, pp. 186–195.

[30] ——, "Using real-time queueing theory to control lateness in real-time systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 25, no. 1, pp. 158–168, 1997.

[31] R. J. Boucherie and N. M. Van Dijk, *Queueing networks: a fundamental approach*. Springer Science & Business Media, 2010, vol. 154.

[32] D. G. Kendall, "Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded markov chain," *The Annals of Mathematical Statistics*, pp. 338–354, 1953.

[33] M. Eenige, van, "Queueing systems with periodic service," Ph.D. dissertation, Department of Mathematics and Computer Science, 1996.

[34] T. J. Ott, "On the stationary waiting-time distribution in the GI/G/1 queue, i: transform methods and almost-phase-type distributions," *Advances in applied probability*, vol. 19, no. 1, pp. 240–265, 1987.

[35] ——, "The single-server queue with independent GI/G and M/G input streams," *Advances in applied probability*, vol. 19, no. 1, pp. 266–286, 1987.

[36] İ. Şahin and U. N. Bhat, "A stochastic system with scheduled secondary inputs," *Operations Research*, vol. 19, no. 2, pp. 436–446, 1971.

[37] G. F. Newell, "Statistical analysis of the flow of highway traffic through a signalized intersection," *Quarterly of Applied Mathematics*, vol. 13, no. 4, pp. 353–369, 1956.

[38] A. K. Erlang, "The theory of probabilities and telephone conversations," *Nyt. Tidsskr. Mat. Ser. B*, vol. 20, pp. 33–39, 1909.

[39] H. Kim, S. Wang, and R. Rajkumar, "vmpcp: A synchronization framework for multi-core virtual machines," in *Real-Time Systems Symposium (RTSS), 2014 IEEE*. IEEE, 2014, pp. 86–95.

[40] S. Karlin, *A first course in stochastic processes*. Academic press, 2014.

[41] N. M. van Dijk, "On the arrival theorem for communication networks," *Comput. Netw. ISDN Syst.*, vol. 25, no. 10, pp. 1135–1142, May 1993.

[42] R. Larson and B. H. Edwards, *Calculus*. Cengage Learning, 2009.

[43] H. Kim and R. Rajkumar, "Real-time cache management for multi-core virtualization," in *Embedded Software (EMSOFT)*, Oct 2016, pp. 1–10.

[44] M. Xu, L. T. X. Phan, H. Y. Choi, and I. Lee, "vcat: Dynamic cache management using cat virtualization," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2017.

[45] "5 tips for running Redis over AWS," https://redislabs.com/blog/5-tips-for-running-redis-over-aws/, 2018.