When Power-of-d-Choices Meets Priority

Jianyu Niu*¶, Chunpu Wang†, Chen Feng‡, Hong Xu§
*Research Institute of Trustworthy Autonomous Systems, Southern University of Science and Technology
†Unity Technologies, ‡School of Engineering, University of British Columbia
§Department of Computer Science and Engineering, Chinese University of Hong Kong
Email: *niujy@sustech.edu.cn, †chunpu.wang@alumni.ubc.ca, ‡chen.feng@ubc.ca, §hongxu@cuhk.edu.hk

***Abstract*—Power-of-d-choices (Pod) is a popular load balancing strategy, which has received much attention from both academia and industry. However, much prior work on Pod has focused on uniform tasks without priorities. In reality, tasks may have different priorities according to their service sensitivity, pricing, or importance to guarantee the quality of service (QoS). In this work, we distinguish two types of priorities in Pod: scheduling and service priorities. We propose Pod-SSP, which is a Pod algorithm with Scheduling and Service Priorities. To better understand the impact of priorities on the performance of tasks, we consider two simple variants of Pod-SSP: Pod with SCheduling Priorities (Pod-SCP) and Pod with SErvice Priorities (Pod-SEP). Utilizing mean-field approximation, we systematically study the performance of these protocols in the large-system regime. Our theoretical and simulation results show that high-priority tasks can have a more than 3x better delay relative to a system running the original Pod algorithm, and meanwhile, low-priority tasks only slightly sacrifice their delay.**

***Index Terms*—Load balance, Power-of-d-Choices (Pod), Task priorities, Quality-of-Service (QoS).**

I. Introduction

Load balancing plays a crucial role in large server farms (e.g., data centers or fog computing) for scheduling incoming tasks amongst servers in order to minimize task response time and improve user experience. Join-the-Shortest-Queue (JSQ) is a natural load balancing algorithm, which tracks the queue lengths of all servers and selects the least loaded server whenever a task arrives. Although JSQ is proven to be delay-optimal in certain regimes [1], it does not scale well. This is because the algorithm needs to continuously monitor servers' loads, which incurs high message and computation overhead.

To improve scalability, a new load balancing algorithm called Power-of-d-choices (Pod) (also known as JSQ(d) in the literature) is proposed with low message overhead [2], [3]. Specifically, Pod probes only d servers uniformly at random upon task arrival and then dispatches the task to the least loaded one among the d samples, as shown in Fig. 1. Despite its simplicity, Pod can achieve a low average task response time even with d as small as two [2]. The salient performance makes Pod gain considerable attention and be widely implemented [4]–[8]. Industry, such as NGINX [4] and HAProxy [5], has already adopted Pod as one of their load balancing algorithms. Besides, Pod has shown great potentials in scheduling applications in various systems, ranging from data center networks [9], [10], large-scale storage systems [6], [11], fog computing systems [7], blockchain systems [12], [13], to wireless networks [8].

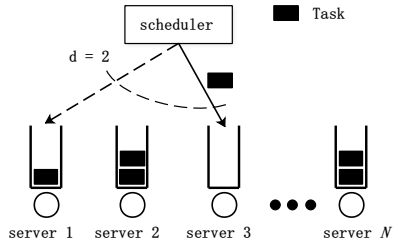


Fig. 1. An overview of the Pod algorithm for many-server system.

However, most prior work on Pod focused on uniform tasks without priorities. In reality, tasks may have different priorities according to their service sensitivity, pricing, or importance. For example, the completion-times of some tasks (e.g., online gaming or real-time search) are more sensitive for users; some users are willing to pay more for their tasks in a system with differential service; certain tasks are more important for business or computing jobs (e.g., a Map job in the Map/Reduce framework). Under these circumstances, the time-sensitive, high-priced, or important tasks need higher priority in execution to guarantee the quality of service (QoS) [14]. Therefore, in order to meet these demands, Pod-based systems should provide such task priorities. This motivates our study.

In this paper, we conduct an in-depth analysis of the Pod algorithm with task priorities. We distinguish two types of priorities: scheduling and service priorities. Algorithms can schedule high-priority tasks to the less-loaded servers by probing more servers, and a server can first process high-priority tasks in its queues. In particular, we propose a new Pod algorithm with **S**cheduling and **S**ervice **P**riorities, namely Pod-SSP. We use the mean-field analysis (also referred to as fluid-limit analysis) to rigorously analyze task response times. Our studies show that high-priority tasks' response times can be significantly shortened with a slight sacrifice of low-priority tasks' delay under some conditions. Besides, we systematically evaluate the impact of these two priorities on tasks' performance by studying two variants of Pod-SSP. Although the focus of this paper is on the Pod algorithm, we believe that the algorithm design and analysis can be extended to other scheduling strategies such as Join-the-Idle-Queue (JIQ) [3], [15].

¶ Part of this work was done when Jianyu Niu was at UBC.

Our main contributions are as follows:

- We propose a new Pod algorithm with **S**cheduling and **S**ervice **P**riorities (Pod-SSP). We study its performance in the large-system regime using a mean-field analysis. The analysis shows that high-priority tasks' response times can be significantly shortened at the only slight sacrifice of low-priority tasks.
- To better evaluate the impact of priorities on tasks' performance, we propose two variants of Pod-SSP: Pod with **SC**heduling **P**riorities (Pod-SCP) and Pod with **SE**vice **P**riorities (Pod-SEP). By studying them, we find that service priorities are more important than scheduling priorities in guaranteeing high-priority tasks' QoS, especially when high-priority workloads are small.
- We perform extensive simulations to verify that our analytical results are indeed accurate in large, but finite, systems. Simulation results show that when workloads are high, even sampling one more server for a small fraction of high-priority tasks can greatly reduce task responses times.
- We discuss communication costs caused by service and scheduling priorities. We also discuss how to achieve zero delay for high-priority tasks in Pod-SSP.

The rest of this paper is organized as follows. In Section II, we provide the related work of the Pod algorithm. We then introduce the system model, the algorithm Pod-SSP, and the main results in Section III. We present the analysis of Pod-SSP in Section IV. We analyzed two variants of Pod-SSP in Section V. We evaluate these protocols in Section VI. We discuss communications costs and optimization of task response times in Section VII and conclude the paper in Section VIII.

II. Related Work

The Pod algorithm and its variants have been widely studied and applied in today's computer systems [2], [3], [16]–[25]. The Pod algorithm was first studied based on the system model of Poisson task arrivals, exponential service times, single dispatcher without memory, and homogeneous servers [2], [3]. Later on, many studies were conducted to extend the above system. For example, Bramson et al. extended the exponential service times to general service requirement distributions [16]. In [17], [18], Pod algorithms with heterogeneous servers are studied. Ying et al. studied the Pod algorithm with batch task arrivals and found that sampling slightly more than one queue on average per task can achieve a lower tail response time than the original Pod algorithm [19]. In [20], Serguei and Natalia studied a multiple-dispatcher system, in which tasks are clustered into several classes, and tasks in the same class are accessible to some dispatchers (not all) for scheduling. They obtained simple stability criteria for two particular cases when service rates are either dispatcher-independent or class-independent. Shay et al. studied the system with heterogeneous servers and multiple dispatchers [23]. In [21], [22], the dispatcher is assumed to have the memory to store servers' queue information such that the dispatcher doesn't need to probe servers every time upon a task arrival. Kristen et al. studied a Pod algorithm, in which an arriving task is copied and dispatched to multiple servers, and the task is completed once any copy is finished [24]. In [25], Wang et al. proposed a hybrid algorithm that combines the Pod with a centralized helper.

Despite the numerous variants of Pod algorithms, there are very few works focusing on task priorities. Mitzenmacher et al. in [26] proposed a simple variant of Pod, which provides scheduling priorities by choosing the shortest server of two servers for high-priority tasks and randomly choosing a server for low priority tasks. This algorithm can be viewed as a special case of our Pod-SCP algorithm. In addition to scheduling priorities, we have also considered service priorities and provided a systematical analysis of both of them in this paper. In [27], Alistarh et al. used the power-of-two-choices to remove elements from multiple concurrent priority queues, and their study is orthogonal to ours. Our work is among the first to systematically analyze the impact of priorities on tasks' performance.

III. System Model and Main Results

A. System Model

We consider an online service with N identical servers, each with a separate queue to store tasks including the one in service, as illustrated in Fig. 2. We assume that task arrivals follow a Poisson process with rate λN. There are two types of tasks, namely, low-priority tasks and high-priority tasks. High-priority tasks enjoy priorities in terms of scheduling and service (with details introduced shortly). The probability that a task belongs to the high priority (resp., low priority) is p (resp., $1-p$). Therefore, p represents the ratio of high-priority tasks and is called the *privilege ratio*. The processing time of each task is exponentially distributed with mean $\mu = 1$, which is independent across tasks and servers. We assume that $\lambda < 1$, because otherwise, the system is unstable. In addition to servers, there is a centralized scheduler that runs the load balancing algorithm to dispatch tasks to servers. Due to space constraints, in this work, we focus on homogeneous tasks, i.e., tasks with different priorities have the same service distribution. Besides, we do not consider a more general service model.

B. Algorithms

We first describe the original power-of-d-choices (Pod) algorithm and then introduce our Pod-SSP, a new Pod algorithm with scheduling and service priorities.

Power-of-d-Choices [2], [3]: *When a task arrives, the scheduler probes d servers uniformly at random (with replacement) and dispatches the task to a server with the shortest queue. If multiple servers have the same shortest queue length, the scheduler chooses one uniformly at random.*

Pod with Scheduling and Service Priorities: *On the server side, each server gives strict priority to high-priority tasks. In particular, a server is processing a low-priority task only if there is no cached high-priority task in its queue. When a*

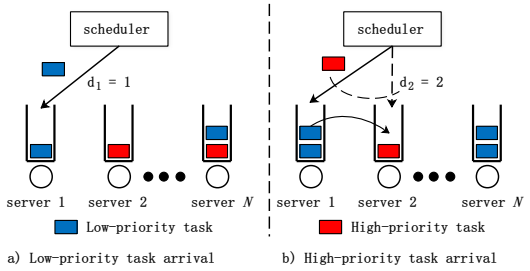


Fig. 2. An illustration of the Pod-SSP algorithm for scheduling tasks.

high-priority task arrives, this server will stop processing its low-priority task (if any) and start processing the new high-priority task.

Note that the above service strategy is called *preemptive resume priority* in the field of priority queuing [28]. When implemented, each server can maintain two separate queues, one for each priority class. When a server becomes free, the first task of the highest nonempty priority queue enters the service. The service of a low-priority task is interrupted when a high-priority task arrives and is resumed from the point of interruption once all high-priority tasks (in the higher priority queue of a server) have been served.

On the scheduler side, upon a low-priority task arrival, the scheduler probes d_1 servers (with replacement) and dispatches the task to a server with the minimum number of all tasks. Upon a high-priority task arrival, it probes d_2 (where $d_2 > d_1$) servers and dispatches the task to a server with the minimum number of high-priority tasks. We call this server s_A. In addition, if server s_A isn't one of the servers with the minimum number of tasks (among d_2 servers), the last low-priority task in s_A is transferred to one of the servers with the minimum number of tasks for better load balancing. (This action happens only if a high-priority task arrives.)

Fig. 1 a) illustrates a simple case in which the scheduler dispatches the low-priority tasks to a randomly chosen server (i.e., $d_1 = 1$). In Fig. 1 b), upon a high-priority task arrival, server 1 and server 2 are randomly chosen (i.e., $d_2 = 2$), and this task is dispatched to server 1 for it has the minimum number of high-priority tasks (i.e., no high-priority tasks in queue). Since server 1 is not the server with the minimum number of all tasks, the last low-priority task in server 1 will be transferred to server 2 by the above algorithm. It is easy to see that the transfer of low-priority tasks makes loads among servers more balanced.

C. Main Results

We first introduce several metrics to characterize the performance of load balancing algorithms and then present our main results.

Task response time. The response time of a task is the time duration between its arrival and departure. The task response time is defined as the average response time of tasks of the same type over the long run. Low-priority and high-priority task response times in Pod-SSP are denoted by $T_L(\lambda, p)$ and $T_H(\lambda, p)$, respectively. The task response time reflects the average delay performance of tasks.

TABLE I
THE EXPECTED RESPONSE TIMES OF TASKS.

$\widehat{T}(\lambda)$	$T_H(\lambda, p)$	$T_L(\lambda, p)$
$\sum_{i=1}^{\infty} \lambda^{\frac{d^i-d}{d-1}}$	$\sum_{i=1}^{\infty} (p\lambda)^{\frac{d^i-d}{d-1}}$	$\sum_{i=1}^{\infty} \left(\frac{1}{(1-p)}\lambda^{\frac{d^i-d}{d-1}} - \frac{p}{(1-p)}(p\lambda)^{\frac{d^i-d}{d-1}}\right)$

Priority gain and loss. We introduce the priority gain and loss to evaluate the impact of priorities on the delay performance of tasks with different priority levels. Specifically, we use $\widehat{T}(\lambda)$ to denote the task response time of the system running the Pod algorithm with $d = d_1$ [2]. (The system can also be treated as a system running Pod-SSP with $p = 0$. Therefore we have $\widehat{T}(\lambda) = T_L(\lambda, 0)$.) The priority gain for high-priority tasks $\gamma(\lambda, p)$ is defined as:

$$\gamma(\lambda, p) = \widehat{T}(\lambda)/T_H(\lambda, p). \tag{1}$$

Similarly, the priority loss for low-priority tasks is defined as

$$\eta(\lambda, p) = T_L(\lambda, p)/\widehat{T}(\lambda). \tag{2}$$

For example, if the high-priority task response time $T_H(\lambda, p)$ is half of $\widehat{T}(\lambda)$, then the priority gain is 2 (indicating a two-fold improvement for high-priority tasks). Clearly, both $\gamma(\lambda, p)$ and $\eta(\lambda, p)$ are greater than one when $p \in (0, 1)$.

In this paper, we study Pod-SSP in the large-system limit (i.e., $n \to \infty$), since a data center today may consist of tens of thousands of servers. Table I presents our main results about $T_H(\lambda, p)$ and $T_L(\lambda, p)$ in Pod-SSP with $d_1 = d_2 = d$. (Note that the expression for $\widehat{T}(\lambda)$ has been provided in Theorem 1 in [2].)

Several remarks are in order. First, we observe that $T_H(\lambda, p)$ is always smaller than $\widehat{T}(\lambda)$ for $p \in (0, 1)$, because $T_H(\lambda, p)$ has smaller base $p\lambda$ (instead of λ) given the same exponent. Hence, the smaller p is, the smaller $T_H(\lambda, p)$ is. Second, we observe that $T_L(\lambda, p)$ is always larger than $\widehat{T}(\lambda)$ for $p \in (0, 1)$. This is because $T_L(\lambda, p)$ can also be expressed as

$$T_L(\lambda, p) = \sum_{i=1}^{\infty} \left(\lambda^{\frac{d^i-d}{d-1}} + \frac{p}{1-p}\left(\lambda^{\frac{d^i-d}{d-1}} - (p\lambda)^{\frac{d^i-d}{d-1}}\right)\right).$$

Clearly, the term $\frac{p}{1-p}\left(\lambda^{\frac{d^i-d}{d-1}} - (p\lambda)^{\frac{d^i-d}{d-1}}\right)$ is always greater than zero for $p \in (0, 1)$. Furthermore, with $T_H(\lambda, p)$, $T_L(\lambda, p)$ and $T(\lambda)$, it is easy to derive $\gamma(\lambda, p)$ and $\eta(\lambda, p)$ by Eq. (1) and Eq. (2), respectively. To better illustrate these results, we provide a concrete numerical example here. When $d_1 = d_2 = 2$, $\lambda = 0.95$ and $p = 0.3$, $T_L(\lambda, p) = 1.08$, $T_H(\lambda, p) = 4.37$, and $\widehat{T}(\lambda) = 3.38$. Additionally, we have $\gamma(\lambda, p) = 3.13$, and $\eta(\lambda, p) = 1.29$. Due to space constraints, results for $T_H(\lambda, p)$ and $T_L(\lambda, p)$ in Pod-SSP with $d_1 \neq d_2$ are provided in Theorem 3 and 4.

IV. Analysis of Pod-SSP

In this section, we prove our main results by studying the delay performance of high-priority and low-priority tasks in Pod-SSP. In particular, we derive the stationary distribution of a single server's queue length under the large-system limit, which allows us to characterize the average task response times.

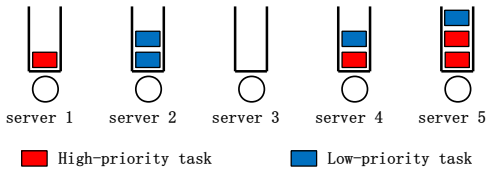


Fig. 3. An illustration of $N = 5$ servers' states at time t.

A. System States

First, we look at the state of a single server in the system. Let $Q_k(t)$ (resp. $\tilde{Q}_k(t)$) for $k \in \{1, 2, ..., N\}$ denote the the queue length of all tasks (resp., high-priority tasks) of server k at time t. For any fixed N, the queue length vector $\{Q_k(t)\}_{k=1}^N$ forms an irreducible, aperiodic, continuous-time Markov chain since all events (task arrivals and departures) are generated according to independent Poisson process, and the change of queue lengths only depends on the current queue lengths. Therefore, $\{Q_k(t)\}_{k=1}^N$ is positive recurrent [2]. Additionally, let $V_i^N(t)$ denote the fraction of servers with at least i tasks at time t, which is also known as the tail distribution. Thus, we have

$$V_i^N(t) = \frac{1}{N}\sum_{k=1}^{N} \mathbf{I}(Q_k(t) \geq i), \quad i \geq 0, \tag{3}$$

where $\mathbf{I}(\cdot)$ is the indicator function. Here, $V_i^N(t)$ can also be interpreted as the tail probability of a typical server having at least i tasks. Also, $\sum_{i=1}^N V_i^N(t)$ is equal to the average queue length of all tasks at time t. We can use the new random process $\{V_i^N(t)\}_{i=1}^\infty$ to represent the server states of all tasks at time t. In fact, $\{V_i^N(t)\}$ is also a Markov process. Similarly, we use $\tilde{V}_i^N(t)$ to denote the fraction of servers with at least i high-priority tasks and the random process $\{\tilde{V}_i^N(t)\}_{i=1}^\infty$ to represent the server states of high-priority tasks at time t.

To better understand these states, we provide a simple example in Fig. 3 in which there are $N = 5$ servers. As server 4 has one high-priority task and one low-priority task at time t, we have $Q_4(t) = 2$ and $\tilde{Q}_4(t) = 1$. Additionally, there are three servers with at least 2 tasks at time t (i.e., server 2, 4 and 5), so $V_2^N(t) = 3/5$. Similarly, we have $\tilde{V}_2^N(t) = 1/5$.

B. Analysis

We start from a special case of Pod-SSP in which $d_1 = d_2 = d$. We then extend to the general case $d_1 \neq d_2$.

Pod-SSP with d. To facilitate our analysis, we consider two other systems that run the Pod algorithm (using d) with the task arrival rates λN and $p\lambda N$, respectively. In addition, we use S, S_1, and S_2 to denote the system running Pod-SSP with d and the other two systems running Pod, respectively. The number of servers and servers' task processing rates are the same in these three systems. Similarly, we use $X_k(t)$ and $Y_k(t)$ to denote servers' queue lengths of tasks in the system S_1 and S_2, respectively. The queue length vectors $\{X_k(t)\}_{k=1}^N$ and $\{Y_k(t)\}_{k=1}^N$ are also two Markov processes. What is more, the preemptive resume priority in Pod-SSP does not affect servers' queue lengths of all tasks [29], and $X_k(t)$ is also positive recurrent. Next, we present several technical lemmas.

Lemma 1. *The Markov chain $\{Q_k(t)\}_{k=1}^N$ of Pod-SSP with d has the same stationary distribution as the Markov chain $\{X_k(t)\}_{k=1}^N$.*

Proof: First, $\{Q_k(t)\}_{k=1}^N$ and $\{X_k\}_{i=1}^N$ are positive recurrent, they all have unique stationary distributions. Next, we show that for any given values $\{q_k\}_{k=1}^N$ and $\{x_k\}_{k=1}^N$ at time t with $q_k = x_k$ for $k \in \{1, 2, ..., N\}$, the transition rates for the state evolution of $\{Q_k(t)\}_{k=1}^N$ and $\{X_k(t)\}_{k=1}^N$ are the same. Specifically, there are two state transition rates: task arrival rate with λN and task departure rate with μN.

Given a task departure and a randomly chosen server (whose index is denoted by the variable j), if there are queued tasks in this server, the chosen server completes a task, i.e., $q_j - 1$ in system S or $x_j - 1$ in the system S_1. Therefore, the state evolution of $\{Q_k(t)\}_{k=1}^N$ is the same as that of $\{X_k(t)\}_{k=1}^N$ under task departures.

Given a task arrival and the same d servers chosen for systems S and S_1, the server with the minimum tasks will be the same and its index is denoted by the variable j. On the one hand, by the Pod algorithm in the system S_1, the j-th server receives the task and its queue length will increase by one, i.e., $x_j + 1$. Meanwhile, other servers still have the same queue length. On the other hand, as there are two types of tasks in system S, we consider two cases.

- High-priority task: According to the Pod-SSP algorithm, this task will be dispatched to the server with the minimum number of tasks (whose index is k). If $j = k$, we have q_j+1. Otherwise, the task is sent to k-th server, and meanwhile, the latest low-priority task from the k-th server will be transferred to the j-th server. Thus, the queue length of the j-th server will be increased by one, i.e., $q_j + 1$, while the queue lengths of other servers remain the same.
- Low-priority task: This task will be dispatched to the server with the minimum number of tasks (whose index is j). So the queue length of the j-th server will be increased by one.

In either case, we can see that the state evolution of $\{Q_k(t)\}_{k=1}^N$ and $\{X_k(t)\}_{k=1}^N$ are the same under task arrivals. Therefore, $\{Q_k(t)\}_{k=1}^N$ and $\{X_k\}_{k=1}^N$ will have the same stationary distribution. ■

Lemma 2. *The Markov chain $\{\tilde{Q}_k(t)\}_{k=1}^N$ of Pod-SSP with d has the same stationary distribution as the Markov chain $\{Y_k(t)\}_{k=1}^N$.*

Proof: The proof is very similar to Lemma 1. Thus, we omit it here due to space constraints. ■

With these two lemmas, we can derive the tail distribution of tasks and expected task response times in Pod-SSP.

Lemma 3. *In the large-system limit, the stationary tail distribution of servers with all tasks and the stationary tail distribution of servers with high-priority tasks in Pod-SSP with d are $v_i = \lambda^{\frac{d^i-1}{d-1}}$ and $\tilde{v}_i = (p\lambda)^{\frac{d^i-1}{d-1}}$, respectively.*

Proof: Given the stationary distribution of $\{Q_k(t)\}_{k=1}^N$, it is easy to derive the corresponding stationary tail distribution of servers by Eq. (3). This also holds for the Markov chain

$\{X_k(t)\}_{k=1}^N$. Additionally, Lemma 1 shows that $\{Q_k(t)\}_{k=1}^N$ has the same unique stationary distribution as the Markov chain $\{X_k(t)\}_{k=1}^N$. Thus, the corresponding stationary tail distributions of the system S and S_1 are the same. Furthermore, by Lemma 2 [2], the stationary tail distribution of servers with tasks can be derived as $v_i = \lambda^{\frac{d^i-1}{d-1}}$. Similarly, by Lemma 2, we can have the stationary tail distribution of servers with high-priority tasks as $\tilde{v}_i = (p\lambda)^{\frac{d^i-1}{d-1}}$. ■

Theorem 1. *In the large-system limit, the expected response time of all tasks and the expected response time of high-priority tasks in Pod-SSP with d are $T(\lambda,p) = \sum_{i=1}^{\infty} \lambda^{\frac{d^i-d}{d-1}}$ and $T_H(\lambda,p) = \sum_{i=1}^{\infty} (p\lambda)^{\frac{d^i-d}{d-1}}$, respectively.*

Proof: In Pod-SSP, a task that arrives at time t becomes the i-th task in the queue with probability $v_{i-1}(t)^d - v_i(t)^d$. Hence, the expected response time of a task in the system is

$$T(\lambda,p) = \sum_{i=1}^{\infty} i\left(v_{i-1}(t)^d - v_i(t)^d\right) = \sum_{i=0}^{\infty} v_i(t)^d.$$

Note that the expected processing time of task is 1. As $t \to \infty$, the limiting system converges to the fixed point. Hence, the expected response times of a task can be made arbitrarily close to

$$T(\lambda,p) = \sum_{i=0}^{\infty} v_i^{d_1} = \sum_{i=1}^{\infty} \lambda^{\frac{d^i-d}{d-1}}.$$

Similarly, we can obtain the expected response time of high-priority tasks. ■

By this theorem, we find that $T(\lambda,p)$ is the same with $\widehat{T}(\lambda)$ (see Table I). This means that the task priorities do not affect the average response time of all tasks.

Theorem 2. *In the large-system limit, the expected response time of low-priority tasks in Pod-SSP with d is $T_L(\lambda,p) = \sum_{i=1}^{\infty}\left(\frac{1}{1-p}\lambda^{\frac{d^i-d}{d-1}} - \frac{p}{1-p}(p\lambda)^{\frac{d^i-d}{d-1}}\right)$.*

Proof: As we know, $T(\lambda,p)$ can also be derived as

$$T(\lambda,p) = pT_H(\lambda,p) + (1-p)T_L(\lambda,p). \tag{4}$$

By using $T(\lambda,p)$ and $T_H(\lambda,p)$ provided by Theorem 1, we can easily derive $T_L(\lambda,p)$. ■

Note that dividing both sides of Eq. (4) by $T(\lambda,p)$, we can obtain

$$1 = p\frac{T_H(\lambda,p)}{T(\lambda,p)} + (1-p)\frac{T_L(\lambda,p)}{T(\lambda,p)} = \frac{p}{\gamma(\lambda,p)} + (1-p)\eta(\lambda,p). \tag{5}$$

In the last step, because $T(\lambda,p) = \widehat{T}(\lambda)$ (see Theorem 1), $T_H(\lambda,p)/T(\lambda,p) = T_H(\lambda,p)/\widehat{T}(\lambda) = \gamma(\lambda,p)$. This equation illustrates the relationship between $\gamma(\lambda,p)$ and $\eta(\lambda,p)$.

Pod-SSP with d_1 and d_2. Similarly, to facilitate our analysis, we consider a system in which the scheduler dispatches low-priority tasks to the least loaded server among d_1 sampled servers, whereas dispatches high-priority tasks to the least loaded server among d_2 sampled servers. The running algorithm is also called Pod-SCP (with more details to be presented in Sec. V). In particular, we assume that the task arrival rate for the system is λN. We use S^* and S_3 to denote the system running Pod-SSP with d_1 and d_2 and the systems running Pod-SCP, respectively. The number of servers and servers' task processing rates are the same in these two systems. Similarly, we use $Z_k(t)$ to denote the queue lengths of tasks of servers in the system S_3. The queue length vector $\{Z_k(t)\}_{k=1}^N$ is also a Markov process.

We have the following observation of Pod-SSP. In Pod-SSP, the number of sampling servers d_1 for low-priority tasks doesn't affect the stationary tail distribution of servers with high-priority tasks and response times of high-priority tasks. The reasons behind this observation are two-folds. First, when scheduling high-priority tasks, low-priority tasks are not considered for choosing the server. Second, servers always first process high-priority tasks, so low-priority tasks don't affect the processing of high-priority tasks. With this observation, we have the following theorem.

Theorem 3. *In the large-system limit, the expected response time of high-priority tasks in Pod-SSP with d_1 and d_2 is $T_H(\lambda,p) = \sum_{i=1}^{\infty}(p\lambda)^{\frac{d_2^i-d_2}{d_2-1}}$.*

Proof: It follows directly from Theorem 1 and the above observation. ■

By analyzing Pod-SSP and Pod-SCP, we have the following lemma.

Lemma 4. *The Markov chain $\{Q_k(t)\}_{k=1}^N$ of Pod-SSP with d_1 and d_2 has the same stationary distribution as the Markov chain $\{Z_k(t)\}_{k=1}^N$.*

Proof: The preemptive resume priority does not affect servers' queue length of all tasks, so the proof is very similar to Lemma 1. We omit it here due to space constraints. ■

With this lemma, we then derive the tail distribution of tasks in Pod-SSP and expected task response times.

Theorem 4. *In the large-system limit, the stationary tail distribution of servers with all tasks in Pod-SSP with d_1 and d_2 is given by*

$$\begin{cases} v_1 &= \lambda \\ v_i &= \lambda\left((1-p)v_{i-1}^{d_1} + pv_{i-1}^{d_2}\right) \quad \text{for } i \geq 1. \end{cases} \tag{6}$$

The expected response time of all tasks is $T(\lambda,p) = \sum_{i=0}^{\infty}\left((1-p)v_i^{d_1} + pv_i^{d_2}\right)$.

Proof: Lemma 4 shows that $\{Q_k(t)\}_{k=1}^N$ has the same unique stationary distribution as the Markov chain $\{Z_k(t)\}_{k=1}^N$. Thus, the corresponding stationary tail distributions of the system S^* and S_3 are the same. By Lemma 5, the stationary tail distribution of servers with all tasks can be derived. Moreover, due to the same stationary tail distribution of all tasks, the response time of all tasks in Pod-SSP with

d_1 and d_2 is the same as that in Pod-SCP. By Theorem 6, the response times of all tasks is

$$\begin{aligned} T(\lambda,p) &= pT_H(\lambda,p) + (1-p)T_L(\lambda,p) \\ &= \sum_{i=0}^{\infty}\left((1-p)v_i^{d_1} + pv_i^{d_2}\right). \end{aligned} \tag{7}$$

■

Theorems 3 and 4 present the task response times of high-priority tasks $T_H(\lambda,p)$ and all task $T(\lambda,p)$, respectively. By Eq. (4), we can easily derive $T_L(\lambda,p)$.

V. Variants and Analysis

In Pod-SSP, there are two types of priorities: scheduling and service priorities. Both of them guarantee that high-priority tasks have shorter response times than low-priority tasks. Hence, two questions arise naturally:

- Do scheduling and service priorities play the same role for high-priority tasks' QoS?
- If not, which one is more important?

To answer the above questions, we need to separately study these two priorities to evaluate their impacts. To this end, we propose two simple variants of Pod-SSP: Pod with **SC**heduling **P**riorities (Pod-SCP) and Pod with **SE**vice **P**riorities (Pod-SEP). Pod-SCP differs from Pod-SSP in not providing service priorities, while Pod-SEP treats all tasks equally when scheduling. We introduce these two algorithms below.

Pod with Scheduling Priorities: *In Pod-SCP, the scheduler probes d_1 servers uniformly at random for a low-priority task, whereas probes d_2 (where $d_2 > d_1$) servers for a high-priority task. For any task, the scheduler dispatches it to the least loaded one among all sampled servers. Servers don't have the priority information of tasks and process them according to the arriving sequence.*

Pod with Service Priorities: *In Pod-SEP, the scheduler behaves in the same way as in Pod, i.e., sampling d servers and dispatching the task to the least loaded server. Servers follow the same policy as in Pod-SSP to process tasks.*

Remark 1. *Note that in Pod-SEP, the scheduler could utilize servers' information of high-priority and low-priority tasks for better scheduling (e.g., Pod-SSP with d). However, Pod-SEP uses the simple Pod algorithm to minimize the impact of scheduling.*

A. Analysis of Pod-SCP

We use the same system states in Sec.IV-A to analyze Pod-SCP. In particular, let $V_i^N(t)$ denote the fraction of servers with at least i tasks at time t. The process $\{V_i^N(t)\}_{i=1}^{\infty}$ is a Markov process. Given states $v, v' \in \{V_i^N(t)\}_{i=1}^{\infty}$, the transition rates from state v to v' are

$$R_{v,v'} = \begin{cases} N(v_i - v_{i-1}), & \text{if } v' = v - \frac{1_i}{N} \\ \lambda(1-p)(v_{i-1}^{d_1} - v_i^{d_1}) + \lambda p(v_{i-1}^{d_2} - v_i^{d_2}), & \text{if } v' = v + \frac{1_i}{N} \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

where we define $v_0 \equiv 1$ for convenience. Now consider a sufficiently small time interval δ. According to the transition rates above and a standard argument for Markov processes, we have

$$\begin{aligned} E[V_i(t+\delta) - V_i(t) | V(t) = v] = \lambda N(v_{i-1}^{d_1} - v_i^{d_1})\delta \\ -N(v_i - v_{i+1})\delta + O(\delta^2), \end{aligned} \tag{9}$$

where $\lambda N(v_{i-1}^{d_1}\delta$ is the probability that during $[t, t+\delta]$, a new task (either low-priority or high-priority task) arrives and is routed to a server with $i-1$ tasks already, and $N(v_i - v_{i+1})\delta$ is the probability that during $[t, t+\delta]$, one of the servers with i tasks completes the task in service. Defining

$$\dot{v_i} = \lim_{\delta \to 0} \frac{E[V_i(t+\delta) - V_i(t) | V(t) = v]}{N\delta}$$

leads to the following mean-field model [2], [3]:

$$\begin{cases} \frac{dv_i}{dt} &= \lambda(1-p)(v_{i-1}^{d_1} - v_i^{d_1}) + \lambda p(v_{i-1}^{d_2} - v_i^{d_2}) \\ & \quad -(v_i - v_{i+1}) \quad \text{for } i \geq 1; \\ v_0 &= 1. \end{cases} \tag{10}$$

The mean-field model is a dynamical system that approximates the original stochastic system by using the expected drift (4) as the system dynamic. We expect the mean-field approximation to be accurate when N is large because each transition leads to only a small change $(1/N)$ in the stochastic system. In that case, the equilibrium point of the mean field model is expected to be "close" to the stationary distribution of the stochastic system. Similarly, by Theorem 1 in [25], we have the following theorem.

Theorem 5. *The Markov process $\{V_i^N(t)\}_{i=1}^{\infty}$ is ergodic, hence it has a unique state steady-state stationary distribution as following*

$$\begin{cases} \lim_{N\to\infty}\lim_{t\to\infty} V_i^N(t) &= v_i \quad \textit{in distribution} \\ \lim_{t\to\infty}\lim_{N\to\infty} V_i^N(t) &= v_i \quad \textit{in distribution.} \end{cases} \tag{11}$$

The proof of this theorem is in a spirit similar to the proof of Theorem 3 in [2] and Theorem 1 in [25]. Here, we are more interested in the fixed point of the steady-state distribution, which can be obtained by solving the above mean-field model as shown in the following lemma.

Lemma 5. *In the large-system limit, Pod-SCP with d_1 and d_2 has a unique fixed point with $\sum_{i=1}^{\infty} v_i < \infty$ given by*

$$\begin{cases} v_1 &= \lambda \\ v_i &= \lambda\left((1-p)v_{i-1}^{d_1} + pv_{i-1}^{d_2}\right) \quad \textit{for } i \geq 1. \end{cases} \tag{12}$$

Proof: It is easy to check that the proposed fixed point satisfies $\frac{ds_i}{dt} = 0$ for all $i \geq 1$. First, $v_1 = \lambda$ at the fixed point can be intuitively obtained by the fact that at the fixed point, the rate at which tasks enter and leave the system must be equal. Then, from the $\frac{ds_i}{dt} = 0$ for all i, we can derive v_i by summing (10) over all $i \geq 1$. (Note that $\sum_{i=1}^{\infty} v_i < \infty$ guarantees that the sum converges absolutely.) ■

There does not appear to be a convenient closed form for the fixed point for v_i. Still, with this lemma, it is easy to

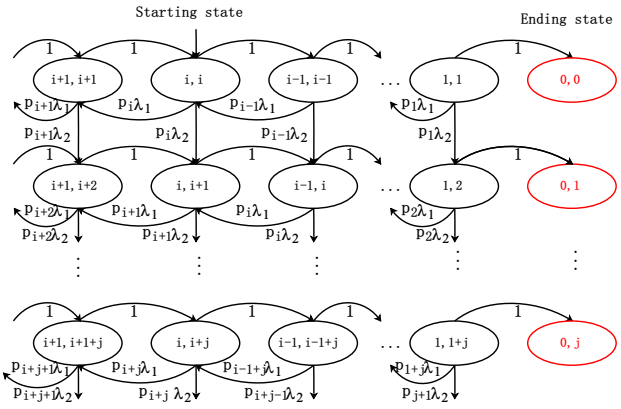


Fig. 4. **The Markov model of a low-priority task becoming the ith task at a server.** Note that $\lambda_1 = \lambda p$ and $\lambda_2 = \lambda(1-p)$. The variable p_i is related with the probability that the server with i tasks is chosen for a new task.

determine the distribution of servers' queue length tasks of tasks, and then derive the expected task response times of tasks in the following theorem.

Theorem 6. *In the large-system limit, the expected response times of high-priority tasks and low-priority tasks in Pod-SCP with d_1 and d_2 converge to $T_L(\lambda, p) = \sum_{i=0}^{\infty} v_i^{d_1}$ and $T_H(\lambda, p) = \sum_{i=0}^{\infty} v_i^{d_2}$, respectively.*

Proof: The proof is very similar to Theorem 1. We omit it here due to space constraints. ■

B. Analysis of Pod-SEP

We also use the same system states in Sec.IV-A to analyze Pod-SEP. As Pod-SEP uses the same scheduling algorithm as Pod, they have the same stationary distributions of servers' queue lengths (of all tasks). Therefore, we have the following lemma by using Lemma 2 in [2].

Lemma 6. *In the large-system limit, Pod-SEP with d has a unique fixed point with $\sum_{i=1}^{\infty} v_i < \infty$ given by $v_i = \lambda^{\frac{d^i-1}{d-1}}$.*

Furthermore, since service priorities don't affect the processing rate of tasks, the expected task response time of tasks in Pod-SEP is the same as that in Pod. Hence, we have the following theorem by Corollary 2 [2].

Theorem 7. *In the large-system limit, the expected response times of tasks $T(\lambda, p)$ in Pod-SCP with $d \geq 2$ converge to*

$$T(\lambda, p) = \sum_{i=1}^{\infty} \lambda^{\frac{d^i-d}{d-1}}.$$

Note that the expected response time of all tasks can be derived by Eq. (4). With $T(\lambda, p)$, we can see that obtaining either $T_L(\lambda, p)$ or $T_H(\lambda, p)$ can easily derive another one. Our observation is that first obtaining $T_L(\lambda, p)$ is easier. This is because, although $T_H(\lambda, p)$ can be easily derived when knowing the stationary distributions of servers' queue lengths of high-priority tasks (see Theorem 1), the stationary distributions of high-priority tasks are coupled with the arrival and processing of low-priority tasks and is hard to be obtained.

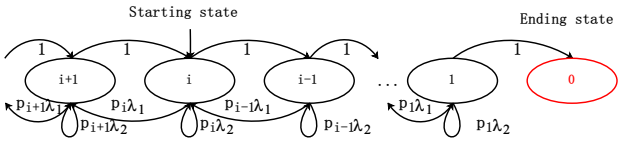


Fig. 5. The simplified Markov model.

Therefore, we turn our focus to the expected task response times of low-priority tasks.

Mean-field approximation. Now consider the queue evolution of one server in the system with a newly arriving low-priority task. Here, we use $P(t)$ and $L(t)$ to denote the queue length of this server and the position of the newly arriving low-priority tasks in this queue. As low-priority tasks are processed after high-priority tasks in Pod-SEP, low-priority tasks are always viewed to be queued after high-priority tasks. It is easy to see that when the low-priority task is arriving at the server, it will be stored at the tail of the queue, and so $L(t) = P(t)$. Besides, we have $L(t) \leq P(t)$.

We can use $(L(t), P(t))$ to capture the state of the low-priority task in the server at time t. Note that we track the state until this low-priority task is completed, i.e., $L(t) = 0$. Fig. 4 illustrates the Markov process of the low-priority task. Given states $(L(t), P(t))$, we can obtain the transition rates for the state evolution. Moreover, by Lemma 6, we can obtain the stationary distributions of N servers' queue lengths in the mean-field analysis. We can further assume that the queue lengths are identically and independently distributed (i.i.d.) with distribution $\{v_i\}_{i=0}^{\infty}$ (which can be validated in the large-system limit). With this, we can derive arrival rates of both the high-priority and low-priority tasks for the server. The results are provided below.

- $q_{(i,j),(i-1,j-1)} = 1$ for $j \geq i \geq 1$.
  This transition happens if the server completes a task in the head of the queue, in which both $L(t)$ and $P(t)$ decrease by one. Therefore, the rate is 1.
- $q_{(i,j),(i,j+1)} = \frac{\lambda(1-p)(v_i^d - v_{i+1}^d)}{v_i - v_{i+1}}$ for $j \geq i \geq 1$.
  This transition happens if a low-priority task is dispatched to the server. The probability for the server with j tasks to be choose is $(v_i^d - v_{i+1}^d)/((v_i - v_{i+1})N)$, where $v_i^d - v_{i+1}^d$ is the probability that a new task is dispatched to servers with ith tasks, and $1/((v_i - v_{i+1})N)$ is the probability that the given server is chosen. The arrival rate of low-priority tasks is $(1-p)\lambda N$. Thus, the transition rate is $\lambda_1 p_j$, where $\lambda_1 = \lambda(1-p)$ and $p_j = (v_i^d - v_{i+1}^d)/(v_i - v_{i+1})$.
- $q_{(i,j),(i+1,j+1)} = \frac{\lambda p(v_i^d - v_{i+1}^d)}{v_i - v_{i+1}}$ for $j \geq i \geq 1$.
  This transition happens if a high-priority task is dispatched to the server. The probability that the server is chosen for a high-priority task is the same as that in the above case. The arrival rate of high-priority tasks is $p\lambda N$. Hence, the transition rate is $\lambda_2 p_j$. Here, $\lambda_2 = p\lambda$.

Next, we will compute the expected time for the low-priority task from state (i, i) $(i \in \{1, 2, 3\})$ to first enter state $(0, j)$ $(j \in \{0, 1, 2, 3\})$. Here, state (i, i) denotes that the task becomes the ith task in the server. However, it is hard to solve the above Markov process. This is because, for each

state, the state transition rate varies according to its queued number of all tasks. To make this solvable, we consider a simple case, in which the arrival of low-priority tasks isn't counted for the number of all tasks. The associated Markov process is illustrated in Fig. 5. In the simple case, the server has fewer tasks than that in reality, and so the arriving rate of high-priority tasks is higher. In other words, more high-priority tasks are inserted before the low-priority task in the queue, and so the waiting time for service is longer. Hence, the derived expected response time is an upper bound for the low-priority task.

We first introduce a new variable r_i to denotes the expected time for a low-priority task from state i ($i \in \{0, 1, 2, 3\}$) to first enter state 0. With the state transitions, we can have the following equation:

$$r_i = \frac{1}{p_i\lambda + 1}(1 + p_i\lambda_2 r_i + r_{i-1} + p_i\lambda_1 r_{i+1}), \text{ for } 1 \leq i. \quad (13)$$

Note that $r_0 = 0$. By defining $d_i = r_i - r_{i-1}$, we have

$$d_{i-1} = 1 + p_{i-1}\lambda_1 d_i, \text{ for } 1 \leq i. \quad (14)$$

Note that when we assume $j+1$ is large, $p_{j+1}\lambda_1$ and $p_{j+1}\lambda_2$ are small. Thus, we have $r_{j+1} = r_j + 1$. By substituting r_{j+1} into Eq. (13), we have

$$r_j = \frac{1}{p_j\lambda + 1}\left(1 + p_j\lambda_2 r_j + r_{j-1} + p_j\lambda_1(r_j + 1)\right).$$

By solving the above equation, we have

$$d_j = 1 + p_j\lambda_1. \quad (15)$$

By substituting d_j into Eq. (14), we can obtain d_{j-1}. Repeating the process will give $d_i = 1 + \sum_{k=1}^{j-i+1} \lambda_1^k \prod_{m=0}^{k-1} p_{i+m}$ ($1 \leq i \leq j$). Finally, we have

$$r_i = \sum_{z=1}^{i} d_z = i + \sum_{z=1}^{i}\sum_{k=1}^{j-z+1} \lambda_1^k \prod_{m=0}^{k-1} p_{z+m}.$$

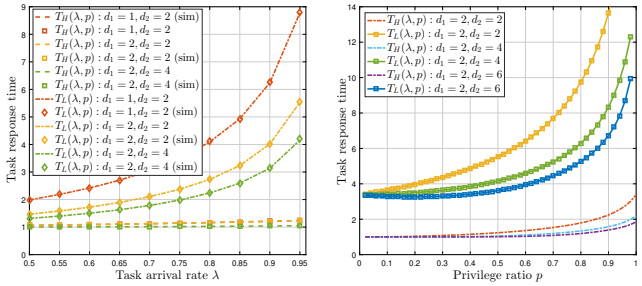
Theorem 8 (Upper bound)**.** *In the large-system limit, the expected response times of low-priority tasks $T_L(\lambda, p)$ in Pod-SEP converge to*

$$T_L(\lambda, p) = \sum_{i=1}^{\infty} r_i(v_{i-1}(t)^d - v_i(t)^d).$$

With Theorem 7 and 8, we can derive a corresponding lower bound for high-priority tasks $T_H(\lambda, p)$.

VI. Evaluation

In this section, we evaluate the performance of Pod-SSP and its variants through simulations. We consider a system with $N = 1000$ servers. The task arrival process is a Poisson process with mean λN, and the task processing time follows an exponential distribution with mean $\mu = 1$ (as described in our system model). We evaluate response times of high-priority and low-priority tasks with different λ, number of sampling servers, privilege ratio p. The simulation results are based on the average of 10 runs, where each run lasts for $100,000$ unit times.



(a) The expected task response time with different λ in Pod-SSP.

(b) The task response times with different privilege ratio p in Pod-SSP.

Fig. 6. The evaluation results of Pod-SSP.

A. Pod-SSP

Fig. 6(a) shows task response times with different λ under $p = 0.5$. We can see that simulation results match the theoretical analysis. From Fig. 6(a), we find that $T_H(\lambda, p)$ slightly increases as λ increases due to its high priorities in scheduling and service. By contrast, increasing λ can significantly affect $T_L(\lambda, p)$, especially when λ is big. This is because due to increasing workloads, low-priority tasks have to wait a long time for being severed until high-priority tasks are completed. Besides, we find that when d_1 is increased from one to two, $T_L(\lambda, p)$ is significantly shortened. This suggests us to set $d_1 \geq 2$ for systems running the Pod-SSP algorithm.

Fig. 6(b) shows task response times with different privilege ratio p under $\lambda = 0.95$ and $d_1 = 2$. We can see that the larger d_2 is, the smaller response times $T_L(\lambda, p)$ and $T_H(\lambda, p)$ are. However, the increase of d_2 only slightly affects the high-priority tasks. This is because high-priority tasks already have shorter delay even under $d_2 = 2$. By contrast, the increasing of d_2 can significantly reduce the $T_L(\lambda, p)$ when p is large. This is because a larger d_2 can make high-priority workloads more balanced, leading to a shorter waiting time for low-priority tasks.

Fig. 7 shows the priority gain $\gamma(\lambda, p)$ and priority loss $\eta(\lambda, p)$ with different privilege ratio p under $\lambda = 0.95$. We can see that the larger p, the larger $\eta(\lambda, p)$ is and the smaller $\gamma(\lambda, p)$ is. Besides, a larger d_2 can guarantee that the increasing high-priority workloads don't affect both $\eta(\lambda, p)$ and $\gamma(\lambda, p)$ too much. This enable us to find a suitable p for the given $\gamma(\lambda, p)$ and $\eta(\lambda, p)$. For example, given a Pod-SSP system with $d_1 = 2$ and $d_2 = 4$, we need to guarantee $\gamma(\lambda, p) \geq 2.5$, $\eta(\lambda, p) \leq 2$, so $\max_p\{\gamma(\lambda, p) \geq 2.5, \eta(\lambda, p) \leq 2\} = 0.5$ can satisfy this.

B. Pod-SCP

Fig. 8(a) shows task response times with different λ under $d_1 = 1$, $d_1 = 2$ and $p = 0.5$. We first see that simulation results match the theoretical analysis. Second, when λ becomes large, both $T_H(\lambda, p)$ and $T_L(\lambda, p)$ suffers from significant increasing. In other words, when workloads are large, a small increase in loads can dramatically increase servers' queue lengths of both low-priority and low-priority tasks.

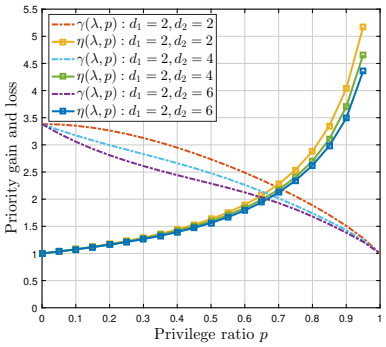


Fig. 7. The priority gain and loss with different privilege ratio p in Pod-SSP.

Fig. 8(b) shows task response times with different privilege ratio p under $\lambda = 0.95$. We can see that when $d_1 = 1$ and $d_2 = 2$, the increasing p ($p < 0.3$) can significantly decrease both $T_H(\lambda, p)$ and $T_L(\lambda, p)$. This means even sampling one more server for a small fraction of high-priority tasks can greatly reduce task responses times. When p is close to one (i.e., two servers are probed for most tasks), the response times become stable. This is, even sampling two servers on average for tasks can significantly decrease task response times. However, when $d_2 \geq d_1 \geq 2$, the increasing of sampling servers d_1 or d_2 neither significantly decreases the response times of tasks nor brings too much advantage for high-priority tasks. This also verifies the previous observations bout Pod-SSP.

C. Task Latency in Pod-SEP

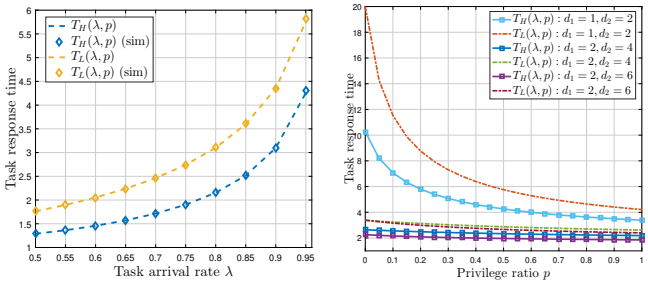
Fig. 8(c) shows task response times with different λ under $d = 2$. From Fig. 8(c), we first observe that when $p = 0.9$, both the theoretical bounds of high-priority and low-priority task response times are close to the simulation results. This is because in this case, the arrival rate of low-priority tasks is low, and so ignorance of them doesn't have too much effect. Similarly, when λ is small (i.e., $\lambda \leq 0.6$), the derived bounds also well match the simulation results. In addition, due to the service priorities, the response times of high-priority low-priority tasks don't increase too much as λ increases.

Fig. 8(d) shows task response times with different privilege ratio p under $\lambda = 0.95$. We can see that the increasing workloads of high-priority tasks (i.e., increasing p) only slightly affect $T_H(\lambda, p)$, whereas $T_L(\lambda, p)$ increases significantly. Besides, by comparing with results in Fig. 8(b), we can see that services priorities play a more important role when p is small.

VII. Discussion

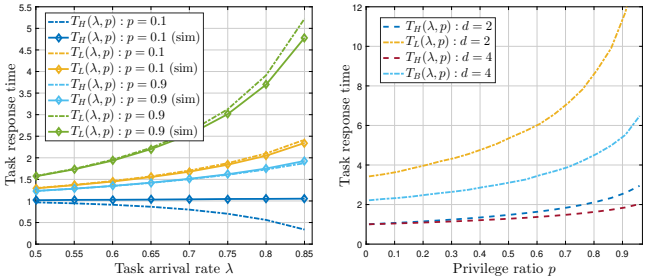
A. Communication Costs

The communication cost is measured as the total number of interactions for a certain amount of tasks. Specifically, an interaction means a probing operation between a scheduler and a server. As analyzed previously, the scheduling priorities (i.e., a larger d) for high-priority tasks can guarantee that they have shorter response times than low-priority tasks. However, this comes at an increasing communication cost.



(a) The task response times with different λ in Pod-SCP.

(b) The task response times with different privilege ratio p in Pod-SCP.



(c) The task response times with different λ in Pod-SEP.

(d) The task response times with different privilege ratio p in Pod-SEP.

Fig. 8. The evaluation results of Pod-SCP and Pod-SEP.

By contrast, service priorities are almost for free. This is because task types could be distinguished by adding one-bit information, and so there are no additional interactions. Besides, in Pod-SSP, the scheduler utilizes servers' reporting information of the number of high-priority and low-priority tasks for better scheduling, which also has no additional communication costs (only additional information fields). What is more, service priorities can play a more important role than scheduling priorities in guaranteeing the QoS of high-priority tasks. Thus, we advise that Pod-SSP should first use service priorities (i.e., $d_1 = d_2$).

B. Achieving Zero Delay for High-priority Tasks

In Pod-SSP, we can choose suitable d_2 for high-priority tasks to achieve zero delay, i.e., an incoming high-priority task is always scheduled to an empty server with a probability asymptotic one. To achieve this, we can set $d_2 = \Omega(\log N/(1-p\lambda))$, when high-priority task arrival rate $p\lambda N$ is less than $(1-\gamma N^{-\alpha})N$ ($0 < \gamma < 1$ and $0 \leq \alpha < 1/6$) [30]. (The α can also be extended to $[0, 1]$ with different settings [30].)

In real systems, task workloads may approach the processing capacity of all servers (i.e., $\lambda N \to \mu N$) for high utilization. Under these cases, it is hard to achieve zero delay for all tasks because of the heavy communication costs (i.e., sampling $\omega(\frac{1}{1-\lambda})$ servers for each task [30]). In addition, this is unnecessary because some tasks may be delay tolerant. By contrast, in Pod-SSP, by categorizing tasks into different priorities according to their QoS demands, we can achieve zero

delay for high-priority tasks using a larger d_1, while achieving low communication costs using a small d_2 for low-priority tasks. In this way, a system can well balance communication costs and tasks' performance.

VIII. Conclusion and Future Work

In this paper, we study the performance of the Pod algorithm with task priorities. We propose Pod-SSP, a new Pod algorithm with scheduling and service priorities. Besides, to better evaluate the impact of priorities on tasks' performance, we propose two variants of Pod-SSP: Pod-SCP and Pod-SEP. We study these three algorithms in the large-system regime using a mean-field analysis. Our studies show that introduced priorities can significantly shorten high-priority tasks' response times, and service priorities are more important than scheduling priorities. We also present how to achieve zero delay for high-priority tasks in Pod-SSP.

There are several avenues for future work. First, in this paper, we only consider exponential service times and homogeneous servers and will extend our analysis to general service and heterogeneous server models in future work. Second, we mostly focused on algorithms with two task priorities. One can extend our analysis to more task priorities. Third, we utilize mean-filed approximation to analyze Pod-SEP. We leave formal proofs as our future work.

IX. Acknowledgement

We thank the anonymous IEEE IWQoS reviewers for their detailed and helpful comments on this paper. We also thank Renming Qi and Zerui Chen for their helpful discussion. Chen Feng was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant RGPIN-2016-05310. Hong Xu also acknowledges the support from the Research Grants Council of Hong Kong (11209520) and CUHK (4055138, 4937007, 4937008, 5501329, 5501517).

References

[1] A. Eryilmaz and R. Srikant, "Asymptotically tight steady-state queue length bounds implied by drift conditions," *Queueing Syst. Theory Appl.*, vol. 72, no. 3–4, p. 311–359, Dec. 2012.

[2] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 10, pp. 1094–1104, 2001.

[3] N. D. Vvedenskaya, R. L. Dobrushin, and F. I. Karpelevich, "Queueing system with selection of the shortest of two queues: An asymptotic approach," *Problemy Peredachi Informatsii*, vol. 32, pp. 20–34, 1996.

[4] O. G. of F5, "NGINX and the "power of two choices" load-balancing algorithm," Nov. 2018. [Online]. Available: https://www.nginx.com/blog/nginx-power-of-two-choices-load-balancing-algorithm/

[5] W. Tarreau, "Test driving, power of two random choices load balancing," Feb. 2019. [Online]. Available: https://www.haproxy.com/blog/power-of-two-load-balancing/

[6] Z. Liu, Z. Bai, Z. Liu, X. Li, C. Kim, V. Braverman, X. Jin, and I. Stoica, "DistCache: Provable load balancing for large-scale storage systems with distributed caching," in *17th USENIX Conference on File and Storage Technologies (FAST)*, Boston, MA, Feb. 2019, pp. 143–157.

[7] X. Gao, X. Huang, S. Bian, Z. Shao, and Y. Yang, "PORA: Predictive offloading and resource allocation in dynamic fog computing systems," *IEEE Internet of Things Journal*, vol. 7, no. 1, pp. 72–87, 2020.

[8] B. Li, J. Liu, and B. Ji, "Low-overhead wireless uplink scheduling for large-scale internet-of-things," *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 577–587, 2021.

[9] P. Wang, G. Trimponias, H. Xu, and Y. Geng, "Luopan: Sampling-based load balancing in data center networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 1, pp. 133–145, 2019.

[10] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, low latency scheduling," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP)*, New York, NY, USA, 2013, p. 69–84.

[11] B. Fan, H. Lim, D. G. Andersen, and M. Kaminsky, "Small cache, big effect: Provable load balancing for randomly partitioned cluster services," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, ser. SOCC '11, 2011.

[12] G. Fanti, J. Jiao, A. Makkuva, S. Oh, R. Rana, and P. Viswanath, "Barracuda: The power of ℓ-polling in proof-of-stake blockchains," p. 351–360, 2019.

[13] B. Doerr, L. A. Goldberg, L. Minder, T. Sauerwald, and C. Scheideler, "Stabilizing consensus with the power of two choices," in *Proceedings of the Twenty-Third Annual ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '11, New York, NY, USA, 2011, p. 149–158.

[14] C. Pham, N. H. Tran, C. T. Do, E.-N. Huh, and C. S. Hong, "Joint consolidation and service-aware load balancing for datacenters," *IEEE Communications Letters*, vol. 20, no. 2, pp. 292–295, 2016.

[15] C. Wang, C. Feng, and J. Cheng, "Distributed join-the-idle-queue for low latency cloud services," *IEEE/ACM Transactions on Networking*, vol. 26, no. 5, pp. 2309–2319, 2018.

[16] M. Bramson, Y. Lu, and B. Prabhakar, "Asymptotic independence of queues under randomized load balancing," *Queueing Syst. Theory Appl.*, vol. 71, no. 3, p. 247–292, Jul. 2012.

[17] A. Mukhopadhyay and R. R. Mazumdar, "Analysis of randomized join-the-shortest-queue (JSQ) schemes in large heterogeneous processor-sharing systems," *IEEE Transactions on Control of Network Systems*, vol. 3, no. 2, pp. 116–126, 2016.

[18] A. Moaddeli, I. N. Ahmadi, and N. Abhar, "The power of d choices in scheduling for data centers with heterogeneous servers," *arXiv preprint arXiv:1904.00447*, 2019.

[19] L. Ying, R. Srikant, and X. Kang, "The power of slightly more than one sample in randomized load balancing," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, 2015, pp. 1131–1139.

[20] S. Foss and N. Chernova, "On the stability of a partially accessible multi-station queue with state-dependent routing," *Queueing Syst. Theory Appl.*, vol. 29, no. 1, p. 55–73, May 1998.

[21] A. Jonatha and D. Francois, "Power-of-d-choices with memory: Fluid limit and optimality," *Mathematics of Operations Research*, vol. 45, 02 2018.

[22] T. Hellemans and B. Van Houdt, "Performance analysis of load balancing policies with memory," in *Proceedings of the 13th EAI International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS)*, New York, NY, USA, 2020, p. 27–34.

[23] S. Vargaftik, I. Keslassy, and A. Orda, "LSQ: Load balancing in large-scale heterogeneous systems with multiple dispatchers," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1186–1198, 2020.

[24] K. Gardner, S. Zbarsky, M. Harchol-Balter, and A. Scheller-Wolf, "The power of d choices for redundancy," *SIGMETRICS Perform. Eval. Rev.*, vol. 44, no. 1, p. 409–410, Jun. 2016.

[25] C. Wang, C. Feng, and J. Cheng, "Randomized load balancing with a helper," in *2017 International Conference on Computing, Networking and Communications (ICNC)*, 2017, pp. 518–524.

[26] A. W. R. Michael, Mitzenmacher, , and R. Sitaraman, "The power of two random choices: A survey of techniques and results," *in Handbook of Randomized Computing*, pp. 255–312, 2000.

[27] D. Alistarh, J. Kopinsky, J. Li, and G. Nadiradze, "The power of choice in priority scheduling," in *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, New York, NY, USA, 2017, p. 283–292.

[28] D. Bertsekas and R. Gallager, *Data Networks (2nd Ed.)*. USA: Prentice-Hall, Inc., 1992.

[29] J. P. Buzen and A. B. Bondi, "The response times of priority classes under preemptive resume in m/m/m queues," *Operations Research*, vol. 31, no. 3, pp. 456–465, 1983.

[30] X. Liu and L. Ying, "On achieving zero delay with power-of-d-choices load balancing," *IEEE Transactions on Network Science and Engineering*, vol. 6, no. 4, pp. 909–916, 2019.