Estimating Multiclass Service Demand Distributions Using Markovian Arrival Processes

RUNAN WANG, GIULIANO CASALE, and ANTONIO FILIERI, Department of Computing, Imperial College London, UK

Building performance models for software services in DevOps is costly and error-prone. Accurate service demand distribution estimation is critical to precisely modeling queueing behaviors and performance prediction. However, current estimation methods focus on capturing the mean service demand, disregarding higher-order moments of the distribution that still can largely affect prediction accuracy. To address this limitation, we propose to estimate higher moments of the service demand distribution for a microservice from monitoring traces. We first generate a closed queueing model to abstract software performance and use it to model the departure process of requests completed by the software service as a Markovian arrival process. This allows formulating the estimation of service demand into an optimization problem, which aims to find the first multiple moments of the service demand distribution that maximize the likelihood of the MAP using generated the measured inter-departure times. We then estimate the service demand distribution for different classes of service with a maximum likelihood algorithm and novel heuristics to mitigate the computational cost of the optimization process for scalability. We apply our method to real traces from a microservice-based application and demonstrate that its estimations lead to greater prediction accuracy than exponential distributions assumed in traditional service demand estimation approaches for software services.

Additional Key Words and Phrases: Service demand distribution, Markovian arrival process, Maximum likelihood estimation, Queueing models, Performance

ACM Reference Format:

Runan Wang, Giuliano Casale, and Antonio Filieri. 2022. Estimating Multiclass Service Demand Distributions Using Markovian Arrival Processes. 1, 1 (November 2022), 25 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Stochastic modeling of microservices is often used in simulation-based and analytical evaluations of system performance. A classic approach to software performance prediction relies on Markovian models, which can help to describe the system analytically and efficiently simulate and forecast performance as needed by both developers and operators. Stochastic models such as queueing networks [15], layered queueing networks [14] and Petri nets [25] have been widely used to model web applications. Similarly, software architecture models are appropriate to describe changes in software components and resources [26], which can be used in conjunction with stochastic models to holistically describe the quality of service in a complex distributed system.

However, choosing the right model parameters is challenging for the predictive accuracy of performance models. In this paper, we focus on *service demands* which are model parameters describing resource consumption for a *single*

Authors' address: Runan Wang; Giuliano Casale; Antonio Filieri, {runan.wang19,g.casale,a.filieri}@imperial.ac.uk, Department of Computing, Imperial College London, London, UK.

Manuscript submitted to ACM

request to a processing unit such as a CPU, a GPU, a disk or other resource.[1] Demands are usually estimated by applying regression methods to measurements of resource utilization and response time collected via system monitoring. Several different approaches for service demand estimation have been proposed over the years, such as utilization law [9], response time approximations based on linear regression [36], non-linear optimization [43], as well as machine learning methods [13].Most of the existing approaches for service demand estimation mainly focus on estimating the *mean* service demand, as opposed to considering the *Service demand distribution*, which is the probability distribution of the demand of time consumption on a specific server without any queueing time. This is different from the response time distribution, which captures the amount of elapsed time from when a request is sent to the time it is completed by the server, including both the service time and the time for queueing. However, as we show in this paper, restricting attention to the mean can limit the accuracy of performance prediction. For example, higher-order moments can affect the predictive accuracy for critical metrics, such as higher percentiles of the response time.

As applications are nowadays updated directly while running in production, to build useful performance models for software services, besides accuracy, it is also important to derive an analytical model directly from run-time data. We focus in this work on microservice-based systems, which are widely used in industry. To learn the service demand distribution, we first model each microservice as a closed queueing system, with the finite closed population representing the maximum parallelism level for served requests due to the microservice admission control. Within this queueing system, the service demand for the queueing station is characterized as an acyclic phase-type (APH) distribution [18, 29]. After generating the continuous-time Markov chain (CTMC) for this model, we filter the departure transitions into a Markovian arrival process (MAP) to characterize the inter-departure times of requests served by the microservice [7, 28]. The problem of service demand estimation can then be formulated as an optimization problem to infer the service demand distribution (i.e., the APH) that maximizes the likelihood of the departure MAP having produced the collected runtime data. In particular, the optimal parameters of the service demand distribution are obtained from matching moments of the APH distribution by a global search with the maximum likelihood estimation procedure.

Since the likelihood function may be non-convex, to overcome the high cost of global optimization, we propose a heuristic estimation method. In this approach, the estimation is divided into sub-problems aimed at fitting different moments. The required measurements for fitting consist of the inter-departure times, response times and the time of departure, which can be directly collected via network traffic sniffing from pairs of arrival and departure events for requests processed at the microservice. To evaluate our method, we validate it against real traces with a single service class, which are obtained from a microservice-based application. The results show that our method can fit the distribution of real traces with a high degree of accuracy.

Based on our previous work on estimating the service demand distribution for a single service class in [42], we then extend our heuristic estimation method to multiclass service processes that can characterize software systems exposing multiple types of services, e.g., the different microservice endpoints. To estimate the service demand for multiclass service queues, we apply class aggregation to simplify the multiclass estimation problem into a sequence of two-class sub-problems that can be used to approximate the initial model. Class aggregation for multiclass queues can effectively help to reduce the computational difficulty of solving CTMCs in the presence of state space explosion. Then, we propose to estimate the service demand distribution for multiclass services with the heuristic method, which is extended to matching the first three moments for different service classes with class aggregation. We also evaluate our

[1]This slightly differs from defining service demands as the cumulative time a request spends receiving service from a resource, accumulated over all visits prior to completion.

extended work on real traces collected from the microservice application with different number of service classes. The experimental results indicate an accurate fit performance of the proposed model prediction to the real traces.

The rest of the paper is organized as follows. Related work is summarized in Section 2. In Section 3, we recall the necessary background and definitions. In Section 4 the inter-departure time model and problem formulation are introduced. In Section 5, we discuss our proposed service demand distribution estimation method based on global optimization with maximum likelihood. In Section 6, we introduce the heuristic method for service demand distribution estimation. We present our experimental results in Section 7. The multiclass service demand distribution estimation method is introduced in Section 8, and the experimental results for multiclass are presented in Section 9. Finally, we draw conclusions in Section 10.

2 Related work

To enable automatic generation for accurate performance modeling, model parameterization brings out an important problem of resource demand estimation. There are several works using regression methods. Rolia *et al.* [38] introduce the resource demand estimation problem with linear regression techniques. In general, linear regression has been employed to solve the service demand estimation, mostly based on utilization [8, 9] and response time [36]. Neural networks like recurrent neural networks (RNN) may also be applied to estimate resource demands for the ability to predict time series data [13]. Machine learning can also help to select the optimal approaches for estimation on account of varieties of existing resource demand approaches [23]. However, most of the mentioned approaches based on regression only enable to obtain the mean value of the resource demand instead of full distributions, lacking higher-order properties of the demand.

There are many examples of works using Poisson arrival processes and exponential service times in system modeling with a Poisson arrival process and an exponentially distributed service time [4, 22]. However, the information captured by the service demand distribution is not leveraged. This information is especially important under First-come First-served (FCFS) scheduling, where mean demands are no longer sufficient to characterize performance exactly as in processor sharing scheduling (PS). To characterize more accurate service demand, PH distributions provide the ability to approximate arbitrary distributions, which has been used in studies of modeling and simulation like [12, 34].

Moment matching is one of the most common methods for approximating PH distributions [24]. This method is mainly based on the optimization techniques like maximum likelihood estimation (MLE) and expectation maximization (EM) algorithm [30, 41] to minimize the difference between the parameters and the PH distribution moments. The selection of the number of moments to use for PH fitting can be arbitrary, while most of the existing works usually take the two or three moments that are sufficient to match a mixture of different distributions, for example, in [1, 31]. Therefore, the study of low order moments is important and common in moment matching methods to fit a PH distribution.

MAPs provide a more general way to model the successive times between event occurrence. MAPs are widely used in the existing works on building traffic models [17, 20] and workload characterization [7], due to the ability to capture the burstiness and correlated traffic features. There are also some analytical studies on the queueing system with MAP arrivals, in which the authors work with MAP/G/K queues [10, 19]. To capture different types of correlated arrivals, Buchholz *et al.* [6] extend single MAP to multiclass MAP with marked arrivals and provide the parameter fitting method. As MAPs are able to incorporate the inter-event times, it is also possible to study the service process with MAPs. However, techniques to infer the departure process from real data using MAPs, to our knowledge, have not received attention in the literature.

3 Preliminaries

Acyclic Phase-type Distribution. A phase-type (PH) distribution [32] is defined as the distribution of absorbing time in a continuous-time Markov chain (CTMC) with finite states $\{1, 2, ..., w, w+1\}$, where the first w states are transient, and the last one is absorbing. The infinitesimal generator matrix of this CTMC is

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{T} & \boldsymbol{t} \\ \boldsymbol{0}^T & 0 \end{bmatrix}$$

The sub-generator $\boldsymbol{T}$ has dimension $w \times w$ and specifies the transition rate from state i to state $j \neq i$ when this does not lead to an absorption. We also define $\boldsymbol{t} = -\boldsymbol{T}\boldsymbol{e}$, where $\boldsymbol{e}$ denotes a column vector of 1 with appropriate dimension. We can further describe the stationary distribution of the transient states with $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_w)$, subject to $\boldsymbol{\alpha}\boldsymbol{e} = 1, \alpha_i \geq 0$. A PH distribution may thus be compactly specified as $PH(\boldsymbol{\alpha}, \boldsymbol{T})$. An acyclic PH (APH) distribution [1] is a subset of PH distributions with an acyclic underlying Markov chain. This implies that any state in the underlying Markov chain cannot be visited more than once before absorption. If a random variable Y has APH distribution with parameter $\boldsymbol{\alpha'}$ and $\boldsymbol{T'}$, we write $Y \sim APH(\boldsymbol{\alpha'}, \boldsymbol{T'})$.

Service demand distribution modeling. In this paper, the service demand distribution is modeled as an APH distribution. This is because Markovian distribution models such as the APH are flexible for moment fitting and composition with CTMCs. Given a set of moments, the parameters of an $APH(\boldsymbol{\alpha}, \boldsymbol{T})$ may be obtained with various PH distribution fitting methods [18]. In this work, we will use the method of moment matching, which can fit the parameters to match an arbitrary number of moments of a reference on empirical distribution, so that the service demand distribution is tackled by finding moments to fit an APH. In particular, we consider using the first three moments to study the APH distribution of the service demand. The third moment (skewness, ν) is considered for its characterization of the fitting performance of the end of the tail

$$\nu = \frac{E[(X-m)^3]}{(E[(X-m)^2])^{3/2}} \tag{1}$$

Here, the random variable X denotes the service demand placed by a request at the resource and its first three moments may be given in terms of the mean (m), the squared coefficient of variance (SCV, c^2) and the skewness (ν), which are related as follows:

$$E[X^2] = (1 + c^2)m^2 \tag{2}$$

$$E[X^3] = \nu(c^2)^{3/2}m^3 + 3m^3c^2 + m^3 \tag{3}$$

Thus, we assume that the service demand distribution may be approximately fitted from knowledge of the first three moments $E[X], E[X^2]$ and $E[X^3]$ which in turn are described by parameters m, c^2, and ν. Our analysis does not rely strongly on the number of moments chosen and may be generalized to situations where more than three moments are used to fit an APH distribution.

Markovian Arrival Processes. PH distribution provides a more general way to capture the service demand distribution, however, it is not always feasible to obtain the measurements from the system that can be directly used for fitting PH-distributed service demand. Here, we consider using MAPs that can reveal the underlying PH process with an explicit counting process.

MAPs [28] are able to incorporate correlations between successive inter-arrival times. An n-state MAP consists of two stochastic processes, referring to a counting process and a phase process modeled by a finite state (n states) CTMC

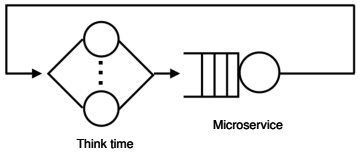


Fig. 1. The reference closed queueing model for a microservice

with infinitesimal generator $\boldsymbol{Q}$. Let $\boldsymbol{D}_0$ be a matrix associated with transitions without arrivals with non-negative off-diagonal elements; $\boldsymbol{D}_0$ and $\boldsymbol{D}_1$ satisfy $\boldsymbol{Q} = \boldsymbol{D}_0 + \boldsymbol{D}_1$ and $(\boldsymbol{D}_0 + \boldsymbol{D}_1)\boldsymbol{e} = \boldsymbol{0}$.

4 Problem formulation

As mentioned, we propose to observe the departure process of a microservice and determine parameters of service demand distribution, modeled as an APH, using maximum likelihood estimation.

To ensure the quality and efficiency of frequent interactions between different services, in microservice-based systems, connection pooling is applied to maintain the number of open connections for the REST-based interactions over HTTP. Hence, each microservice has a finite upper bound on the maximum number of concurrent users. Thus, a closed queueing model can capture the finite parallelism level of microservices.

Moreover, increasing populations of jobs under closed model may be used to approximate open models [3]. Therefore, the proposed method is still applicable in principle to capture open workload-based systems. In this paper, we then abstract the microservice-based applications as closed queueing models as follows. Figure 1 illustrates the analytical model for this service, consisting of a closed queueing network describing both the microservice buffer and serving process, as well as the think time of clients issuing requests. The model features N concurrent users, each modeled as a job. Scheduling could be either first-come first-served (FCFS) or Processor-sharing (PS) order, depending on the implementation details of the admission control system within the microservice. We assume exponentially distributed user think times at the delay station. The problem we study is to determine the APH service demand distribution in the queueing station. Note that since we focus on a single class of jobs, the queueing model admits a product-form solution for the steady-state distribution, while no specific product-form simplifications are available to stochastically analyze the departure process at the queue. As we show next, the service distribution identification problem requires instead a detailed Markovian analysis.

Departure process modeling with MAP. Referring to [2], the inter-event times in queueing models can be captured with a quasi birth-and-death (QBD) process as follows. The state space of a QBD process is the set of $\{(k,l)|1 \le k \le n_l, l \ge 0\}$ can be divided into levels, where k is referred to as the phase and l is the level of the QBD. Transactions are only allowed between adjacent levels in QBD processes. Therefore, the generator matrix $\boldsymbol{Q}$ of such a QBD process is given by the quasi birth-death process (QBD)

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{L}^0 & \boldsymbol{F}^0 & & \\ \boldsymbol{B}^1 & \boldsymbol{L}^1 & \boldsymbol{F}^1 & \\ & \boldsymbol{B}^2 & \boldsymbol{L}^2 & \boldsymbol{F}^2 \\ & & \ddots & \ddots \end{bmatrix}$$

where $\boldsymbol{L}^l$ represents the transition rates within level l, $\boldsymbol{F}^l$ is for the transitions from level l to level $l+1$ and $\boldsymbol{B}^l$ describes the backward transactions from level l to level $l-1$ when $l \ge 1$.

We can generate the infinitesimal generator $\boldsymbol{Q}$ of the underlying CTMC and then filter the events associated with job departures in matrix $\boldsymbol{D}_1$. That is, all and only the transition rates for departures from the queue are included in the non-negative matrix $\boldsymbol{D}_1$. Then, a MAP can be used to model the departure process with representation $(\boldsymbol{D}_0, \boldsymbol{D}_1)$, where $\boldsymbol{D}_0 = \boldsymbol{Q} - \boldsymbol{D}_1$.

We consider a MAP $= \{\boldsymbol{D}_0, \boldsymbol{D}_1\}$ that represents the departure process of the queueing station, our objective is to estimate the parameters for service demand distribution as $APH(\boldsymbol{\alpha}, \boldsymbol{T})$ from the observed inter-departure times (IDTs). We denote the time between two successive departure events i and $i-1$ as $X_i = t_i - t_{i-1}$. Thus, the IDTs of jobs are the set $\boldsymbol{X} = [X_1, X_2, \ldots, X_n]$. Since the departure process is modeled as a MAP, the IDTs follows a PH distribution $PH(\boldsymbol{\pi}, \boldsymbol{D}_0)$, where $\boldsymbol{\pi} = \boldsymbol{\pi}(-\boldsymbol{D}_0)^{-1}\boldsymbol{D}_1$ indicating the stationary distribution of the embedded chain. If $\boldsymbol{D}_0$ is acyclic, then the PH distribution specializes into an APH one. This distribution produces an interval stationary initialization for the MAP.

For the MAP described above, the joint probability density function (PDF) of IDTs $\boldsymbol{X}$ is

$$f(\boldsymbol{X}) = \boldsymbol{\pi} e^{\boldsymbol{D}_0 X_1} \boldsymbol{D}_1 e^{\boldsymbol{D}_0 X_2} \boldsymbol{D}_1 \ldots e^{\boldsymbol{D}_0 X_n} \boldsymbol{D}_1 \boldsymbol{e} \tag{4}$$

For computational convenience, we assume that the given departure events are independent. Thus, we have the approximation

$$\log f(\boldsymbol{X}) = \sum_{i=1}^{n} \log(\boldsymbol{\pi} e^{\boldsymbol{D}_0 X_i} \boldsymbol{D}_1 \boldsymbol{e}) \tag{5}$$

In general, let $\boldsymbol{S}$ be the parameter set of the service demand distribution to be estimated. The log-likelihood for the IDTs is

$$\log f(\boldsymbol{S}|\boldsymbol{X}) = \sum_{i=1}^{n} \log(\boldsymbol{\pi} e^{\boldsymbol{D}_0(\boldsymbol{S}) X_i} \boldsymbol{D}_1(\boldsymbol{S}) \boldsymbol{e}) \tag{6}$$

where $\boldsymbol{D}_0(\boldsymbol{S})$ and $\boldsymbol{D}_1(\boldsymbol{S})$ explicit the functional dependencies between $\boldsymbol{D}_0$, $\boldsymbol{D}_1$ and the service demand distribution parameters $\boldsymbol{S}$ we seek for, i.e., its moments m, c^2, and ν. Then the service demand distribution estimation problem can be formulated as finding the parameters that maximize the log-likelihood of the IDTs measured from the monitoring traces.

Data preprocessing. For a real system, there could be several requests from the users arriving within a very short period of time. If we directly take all the samples in the trace, it could be quite time-consuming to calculate the likelihood function in Equation (6), especially due to the cost of evaluating the matrix exponential.

To address the issue, we observe that the inter-departure times of jobs can be grouped into different patterns. In order to accelerate the execution times, we apply clustering based on k-means [21] to partition the IDTs to obtain K groups of data with cluster centroids $\boldsymbol{Y} = [Y_1, Y_2, \ldots, Y_K]$. Then, the log joint density in Equation (5) may be approximated based on the IDT clusters as

$$\log f(\boldsymbol{Y}) = \sum_{i=1}^{K} l_i \log(\boldsymbol{\pi} e^{\boldsymbol{D}_0 Y_i} \boldsymbol{D}_1 \boldsymbol{e}) \tag{7}$$

where l_i denotes the number of points in cluster i.

We only briefly discuss the clustering of IDTs in this section and give an experimental example of the trade-off between computational complexity and effectiveness is shown in Section 7.2.

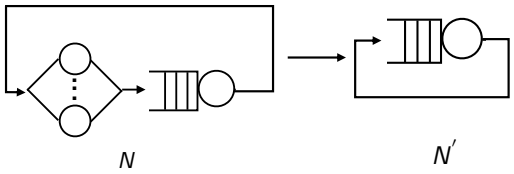


Fig. 2. Modified model for PS with permanent customers perpetually looping at the queueing station

5 Global optimization based estimation

The moment estimation of service demand distribution is based on measurements at runtime. Our measured observation consists of timestamps of arrival and departure for a sample set of successive requests, from which it is easy to derive quantities such as inter-departure times and response time at every resource of interest, we now discuss how we use this data to find a maximum likelihood estimation of the moments. Since an APH can be then recovered from the moments via moment matching, this provides a solution to the service demand distribution fitting problem.

5.1 State space reduction

As we increase the number of concurrent users, the state space of the CTMC can easily suffer state-space explosion. Assume a single-server queue where the service demand SD of the queue node is APH distributed and a delay node as shown in the example in Figure 1. First, we consider the jobs in the queue are processed in a first-come, first-served (FCFS) order. Only one job can be served by the server at one time. Let P denote the number of phases in the service process, i.e., the number of columns in $\boldsymbol{\alpha}$. The number of states in the state space is

$$s = N \cdot P + 1 \tag{8}$$

where N is the number of users in the system.

Instead, if the server follows a processor sharing (PS) scheduling strategy, i.e., multiple jobs can be served simultaneously, and the total number of states is

$$s = \sum_{i=0}^{N} (N + 1 - i) \binom{i + P - 2}{i} \tag{9}$$

To mitigate the complexity of dealing with PS scheduling, we approximate the model as follows. We focus on the mean queue length that can be efficiently estimated using approximate mean value analysis (AMVA) [11]. Based on solving the CTMC, the computational complexity is $O(N^3)$ even for FCFS scheduling. However, AMVA complexity is independent of N. The computational complexity is $O(I)$ for a single class case, where I is the number of iterations required for AMVA to converge, which is drastically fast to obtain the estimated number of permanent users for the queue in milliseconds.

The structure of the modified model is shown in Figure 2. Instead of considering all the individual jobs circulating in the delay and queue nodes as usual, we propose a modified model where the mean number of jobs residing at the queueing station in the original model is forced in the modified model to *permanently* reside at that station with N' *permanent users*, to mimic a similar average congestion level. We approximate N' with

$$N' = \frac{N-1}{N} E[Q(N)] \tag{10}$$

where $E[Q(N)]$ is the expected number of jobs at the queueing station in the original model. Note that the number of jobs N' looping in the new model may be seen as an approximation for the expected number of users at an arrival instant in the queue based on Schweitzer's Approximation [11], which gives the $\frac{N-1}{N}$ term.

For a real system with a large number of users, this approximation can lead to a significant reduction of the computational cost, while providing accurate results.

5.2 Global optimization based estimation

Given the observed trace data, a common approach for parameter estimation is maximum likelihood estimation (MLE) [33], which casts the estimation as a global optimization problem. Thus, we propose an estimation method that combines MLE with simulations of queueing models to approximate the APH distribution of the service demand.

Algorithm 1 describes the implementation of this method in detail. The algorithm requires a set of clustered IDTs and searching boundaries as inputs. We first define a function to calculate the likelihood based on departure MAPs as shown at Line 1 to 16. We use moment matching [1] to fit an APH distribution for the service demand, pointing to *APHFit* in Line 3. Note that we need to satisfy that the APH distribution is feasible with given parameters, i.e., both $\boldsymbol{\alpha}$ and $\boldsymbol{T}$ are nether empty nor zero. After obtaining the service demand distribution SD, a queueing model is generated with a queueing station of SD. The current queueing model can be solved by analyzing the underlying CTMC, obtaining the infinitesimal generator $\boldsymbol{Q}$. By analyzing the transitions in $\boldsymbol{Q}$, the transition rates of departure events on the queue node can be filtered for $\boldsymbol{D}_1$ at in Line 5-7.

However, the computation of the infinitesimal generator involves the evaluation of a matrix exponential, which is computationally expensive and can incur numerical instability [27]. To mitigate these issues, we use CTMC uniformization [37] which is an efficient numerical method for computing transient measures involving matrix exponential. For transient analysis, uniformization techniques can be applied with sub-generator $\boldsymbol{D}_0$ and the initial distribution $\boldsymbol{\pi}$ of the MAP. Since the transient rate in $\boldsymbol{D}_0$ of a real system could be large, to guarantee stable calculations, we adopt the scaling method from [39], involving a scaling factor q to avoid floating-point errors. In Line 9, the scaling CTMC uniformization method is defined as *CTMCUniform*, which takes π, $\boldsymbol{Q} - \boldsymbol{D}_1$ and the centroids of the cluster as the input. The approximated transient probability β can be obtained with $CTMCUniform$. Then the log-likelihood value can be computed using Equation (7) at Line 10. Therefore, we can obtain the optimal moments to fit the service demand distribution by maximizing the likelihood on the monitoring traces at Line 18.

6 Heuristics-based estimation

As illustrated in the last section, the global optimization for service demand distribution estimation needs to consider a large search space. It could be time-consuming to obtain the optimal parameter set maximizing the likelihood value. The method presented in this section estimates the parameters sequentially, rather than jointly, offering a heuristic estimation that trades accuracy for speed.

Mean service demand estimation. The mean value of service demand can be efficiently estimated based on performance measurements from monitoring traces. We refer to the work in [36], which allows estimating the expected value of service demand with queue length and response times. Both queue length and response times are easily measured with system monitoring. As we target on modeling the departure process in the queueing station, the input dataset of the estimation method contains the following data by calculating from system monitoring at departing occurrence.

- The response times from the monitoring traces (R)
- The queue length seen upon arrivals (A)

Considering a single class of jobs in the system, let N be the size of the population in the closed queueing network. It is known that the mean service demand $E[D]$ for the single-class case can be estimated as [36]:

$$E[D] = \frac{E[R]}{1 + E[A]} \tag{11}$$

Algorithm 1 Global optimization based estimation method

Input: Y ← Set of clustered inter-departure times $[Y_1, Y_2, \ldots, Y_n, l_1, l_2, \ldots, l_n]$, where Y_i is the centroid value and l_i is the number of points in cluster i
LB ←moment lower bound
UB ←moment upper bound
Output: SD ← Estimated service demand distribution

function $Likelihood(\mu_1, \mu_2, \mu_3, \boldsymbol{Y})$
  Fit service demand distribution $APH(\boldsymbol{\alpha}, \boldsymbol{T}) \leftarrow APHFit[\mu_1, \mu_2, \mu_3]$
  if $APH(\boldsymbol{\alpha}, \boldsymbol{T})$ is feasible
    Generate a queueing network model with service demand $APH(\boldsymbol{\alpha}, \boldsymbol{T})$
    Solve the generated model and obtain the infinitesimal generator $\boldsymbol{Q}$
    Filter $\boldsymbol{D}_1$ from $\boldsymbol{Q}$
    MAP ← $\{\boldsymbol{Q} - \boldsymbol{D}_1, \boldsymbol{D}_1\}$, generate $\boldsymbol{\pi}$
    for $i = 1$ to n
      $\boldsymbol{\beta} \leftarrow CTMCUniform(\boldsymbol{\pi}, \boldsymbol{Q} - \boldsymbol{D}_1, Y_i)$
      $L \leftarrow L + \log(\boldsymbol{\beta} \boldsymbol{D}_1 \boldsymbol{e}) l_i$
    end for
  else
    $L \leftarrow -\infty$
  end if
  return L
end function
$[\mu_1^*, \mu_2^*, \mu_3^*] = \underset{\mu_1, \mu_2, \mu_3}{\operatorname{argmax}} \, Likelihood(\mu_1, \mu_2, \mu_3, \boldsymbol{Y})$
return $SD = APH(\boldsymbol{\alpha}, \boldsymbol{T}) \leftarrow APHFit[\mu_1^*, \mu_2^*, \mu_3^*]$

where $E[R]$ and $E[A]$ are the expected values of response time and queue length seen upon arrival, respectively.

SCV estimation. To estimate the second moment of service demand, we investigate the state-dependent behavior of the system. The estimation formulation is derived from the SCV on the mean queue length seen upon arrival. In a queueing system, the response time of a job is related to the number of jobs in the queueing station either waiting or receiving service. We define R_i as the response time of the i^{th} request and A_i is the queue length seen upon the arrival of the request i. Then the response times can be then grouped into $max(A)$ sets of data according to the value of A. Let $R^{'}$ be the grouped set of response times (e.g., $\boldsymbol{R}_{\boldsymbol{k}}^{'} = [R_i, \ldots, R_j]$ where $R_i, \ldots, R_j$ have the same queue length A_k with $0 \leq k \leq max(A)$ and $R_{ki}^{'}$ is the i^{th} response time in $R_k^{'}$), resulting in a representation with $(R_k^{'}, A_k)$.

We propose to estimate c^2 using the following heuristic expression

$$c^2 = \max \frac{E[(R_{ki}^{'} - E[R_k^{'}])^2]}{E[R_k^{'}]^2} \tag{12}$$

To demonstrate the accuracy of the approximation for SCV, we conduct an experiment with $N = 100$. We analyze real trace data from system monitoring and compare the c^2 of real traces to the simulated queueing model with estimated parameters based on MLE. We can see from Figure 3(a) that the c^2 conditional on the queue length first increases to the maximum and then decrease with the growth of queue length. The same pattern can also be observed for the real trace as shown in Figure 3(b). The simulated c^2 value is 8.3. It can be seen that the $max(c^2)$ of the simulation is close to the one for the real trace with $c^2 = 11.2$.

Heuristic service demand distribution estimation. Our heuristic method to accelerate the MLE estimation is shown in Algorithm 2. Compared to the global optimization in Algorithm 1, the heuristic-based method is used to estimate the

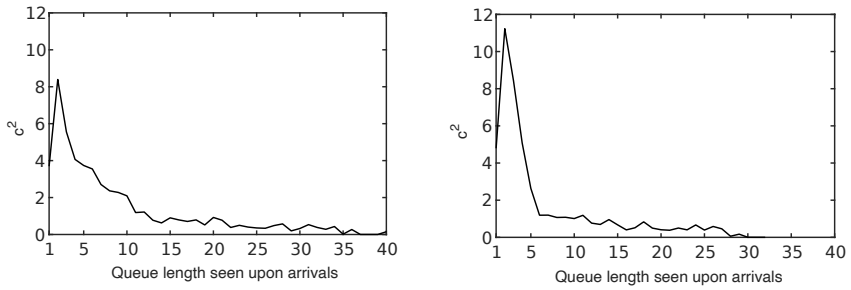


Fig. 3. c^2 conditional on the queue length seen upon arrivals for simulation (a) and the real trace (b)

Algorithm 2 Heuristics based estimation method

Input: $Y \leftarrow$ Set of clustered inter-departure times
$R \leftarrow$ Set of response times $[R_1, R_2, \ldots, R_n]$
$DT \leftarrow$ Set of departure times $[dt_1, dt_2, \ldots, dt_n]$
$\nu = [\nu_1, \nu_2, \ldots, \nu_j] \leftarrow$ searching set of skewness ν
Output: $SD \leftarrow$ Estimated service demand with APH distribution

1: Compute A from DT and R
2: $m \leftarrow meanEstimate(A, R)$
3: $c^2 \leftarrow scvEstimate(A, R)$
4: $\mu_1, \mu_2 \leftarrow m, c^2$ /*Compute the first two moments according to Equations (1) and (2)*/
5: **for** $i = 1$ to j **do**
6: $\quad \mu_3 \leftarrow \nu_i$ /*Compute the third moment based on Equation (3)*/
7: $\quad L \leftarrow Likelihood(\mu_1, \mu_2, \mu_3, \boldsymbol{Y})$ /*Algorithm 1 at Line 2 to 16*/
8: **end for**
9: $\mu_3^* = \underset{\mu_3}{\text{argmax}}\, Likelihood(\mu_1, \mu_2, \mu_3, \boldsymbol{Y})$
10: **return** $SD = APH(\boldsymbol{\alpha}, \boldsymbol{T}) \leftarrow APHFit[\mu_1, \mu_2, \mu_3^*]$

service demand distribution with a series of methods to fit the first three moments, including mean service demand (m), SCV (c^2) and the skewness (ν) estimation.

The algorithm requires a set of IDTs, the departure times, the response time of each job, and the queue lengths seen upon arrivals. In Line 1, we first compute the arrival times with the departure times DT and response times R and then calculate the queue length seen upon arrivals for each job. The mean service demand is estimated based on response times and the queue length seen upon arrivals. Then the algorithm estimates c^2 through Equation (12). The only search parameter for our method now is the skewness ν. Here we perform MLE on the departure process with Equation (6). As in the search-based global optimization, we first generate a set of candidate third moments parameters μ_3 with skewness values $\nu = [\nu_1, \nu_2, \ldots]$. For each ν_i, we generate a queueing model with corresponding service demand and then calculate the likelihood value with function *Likelihood* in Algorithm 1 from Line 1 to 16. The algorithm will search on all candidates in the set, and the process is repeated with multiple candidate points for robustness. The estimated result of the skewness is finally decided by the one that maximizes the likelihood value. Therefore, the final result is obtained by three-parameter estimation methods.

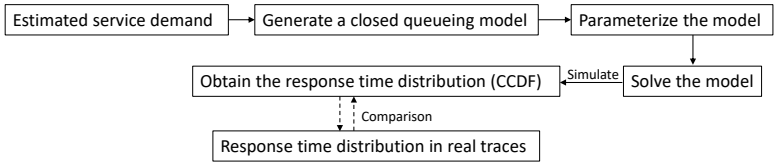


Fig. 4. The process of experimental comparison

Table 1. Workload mixes

CPU level	*Low*	*Medium*	*High*
CPU utilization (U)	33%	43%	95%
Number of users (N)	50	100	300

7 Evaluation

This section introduces the experimental setup and evaluation metrics, together with a comparison of our method against baseline algorithm.

7.1 Experimental setup

To evaluate the proposed service demand distribution estimation, we conduct several experiments using an open-source microservice-based application called Sock Shop[2], which emulates an ecommerce website designed for testing microservices and cloud native technologies. Sock Shop consists of 13 different services, and all services communicate using REST APIs over HTTP. We use Docker Compose for the multi-container orchestration to deploy the microservice-based system. However, in the experiment, we target a service that does not interact with other databases, which avoids indirect drifts in the response time due to the state of the database.

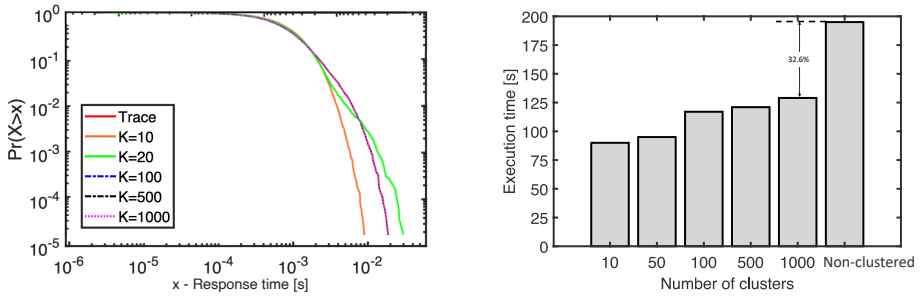
To perform the load test, we then generate closed workloads with a different number of users by using Locust[3]. Locust is an open source load testing tool, which allows emulating microservice users by defining tasks with Python and injecting an artificial workload into the system.

The details of the experimental environment are described as follows. For the deployment of the application, we use a server running Ubuntu 16.04.7, and our target service is pinned to a separate CPU core. Locust is running on 6 different client machines to simulate the concurrent users for the microservice. We experiment with different populations of users to assess the corresponding CPU utilization level as shown in Table 1. In all the experiments, the users' think time is setting with an exponential distribution with a mean value of 0.1 seconds.

First, we run numerical experiments to compare the estimation results to known service demand distribution parameters. In numerical experiments, the trace data is collected by generating of closed queueing models and solve the models with simulation. Next, to evaluate on real microservice traces, we conduct experiments with the workload mixes in Table 1 with Sock Shop and capture the network traffic with a dockerized tcpdump that is triggered over HTTP. During the experiments, we monitor HTTP traffic on the source and destination nodes of our target service. Then the traffic data is parsed to extract the request and response information of each request. For each workload pattern, we collect 10,000 successive HTTP requests and response pairs to build up the trace dataset. Each sample in the trace dataset consists of the timestamps of the request arrival and departure.

Baseline algorithm. The service demand distribution is usually modeled as exponential fitting the estimated mean value [40]. In our experiments, we also fit as a baseline with an exponential distribution for service demand.

[2] https://microservices-demo.github.io/
[3] https://locust.io/



(a) (b)

Fig. 5. The CCDF of response times (a) and the execution times (b) of maximum likelihood estimation under different number of clusters

Metrics. Since the real service times of systems are usually difficult to measure, we opt to use the complementary cumulative distribution function (CCDF) to compare the response time distribution modeling accuracy of the model prediction to the real trace following the methodology as shown in Figure 4. After estimating this service demand distribution, we construct the closed queueing models. Then we use the simulation-based solver JMT in LINE [35] to solve the generated models, and obtain the corresponding response times CCDF from the parameterized model and baseline for comparison with the measure CCDF. The only difference between the baseline and our experiments is the service demand distribution, since the service demand distribution of the baseline is estimated by an exponential distribution with the mean value m.

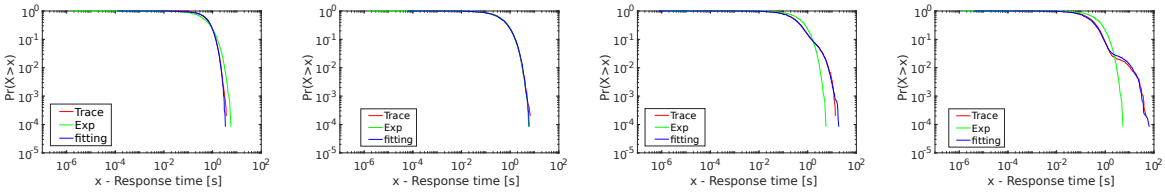
7.2 Data preprocessing and clustering

To demonstrate the trade-off between computational cost and approximation accuracy depending on chosen number of clusters K, we conduct the experiment with the microservice trace when $N = 50$. As mentioned in Section 7.1 each trace dataset consists of 10000 samples, in this experiment, we refer the original data without clustering to $K = 10000$. Then we estimate the service demand distribution with both the original data ($K = 1000$) and clustered data for different values of $K = [10, 50, 100, 500, 1000]$. Figure 5 presents the accuracy and execution time comparison for different K. It can be observed from Figure 5(b) that with clustered IDTs the execution time drops by almost 33%, comparing the results between the original data ($K = 10000$) and $K = 1000$. The accuracy of estimation with the clustered data is evaluated in Figure 5(a) by CCDF diagrams of the response time distributions. While small values of K lead to a coarse approximation, by increasing K, the CCDF for the clustered data rapidly converges to the curve representing the results with the original data (non-clustered). As can be noted from Figure 5(a), the curves become indistinguishable for $K \geq 100$. We can thus conclude that our clustering heuristics does not significantly reduce the accuracy of the estimation when $K \geq 100$, while significantly reducing the computational cost of the estimation.

7.3 Numerical experiment results

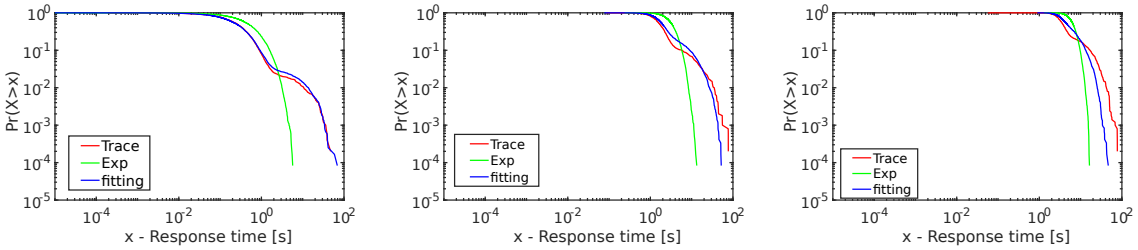
We conduct the following numerical experiments to assess our global-search based method for the service demand distribution estimation, aiming to obtain a ground-truth with pre-setting parameters.

We generate closed queueing networks with a single server queue that can simulate the behaviors of a simplistic microservice, which is denoted as *generated model* in the following description. In the generated model, we give the moments of service demand distribution at the queueing node with different values, so that we can have a ground truth of the moments of service demand distributions. Then we collect samples of timestamps of arrival and departure as the trace data by solving the generated models. In this way, we can compare the estimated results from global optimization to these values setting in the generated model.



(a) $c^2 = 0.5$ (b) $c^2 = 1$ (c) $c^2 = 4$ (d) $c^2 = 16$

Fig. 6. CCDF of response time for different setting of c^2



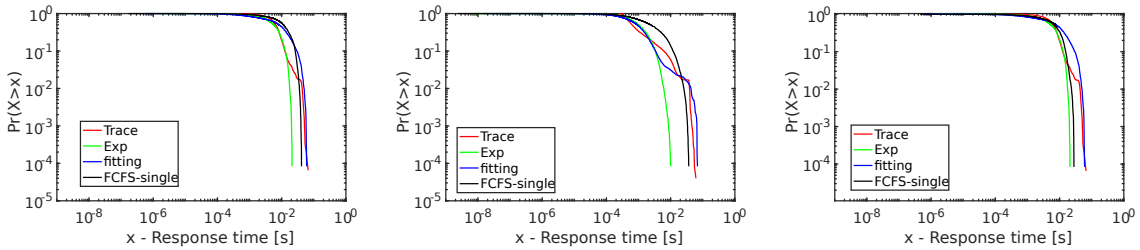
(a) $N = 1$ (b) $N = 5$ (c) $N = 10$

Fig. 7. CCDF of response time for different numbers of users N

The details of the numerical experimental design are as follows. For the setting of pre-known service demand distribution, we fix the mean service demand m with a random value 0.7 and then set different SCV with $c^2 \in \{0.5, 1, 4, 16\}$. Therefore, there are 4 generated queueing models with different service demand distributions. Then we execute Algorithm 1 with these 4 traces data and compare the estimated SD to the known values and the baseline algorithm. The results of the comparison are plotted as CCDF diagrams in Figure 6.

It can be seen from the response times distributions that our estimation method achieves good fits for all settings of c^2 and outperforms the exponential distribution, especially for the tail of the distributions. For larger c^2, such as $c^2 = 16$, we can observe that estimating the mean m of the exponential distribution alone cannot capture the full distribution of service demand, with the baseline algorithm decreasing much faster than our method (e.g., the green curve in Figure 6), resulting in an inaccurate prediction of the response times.

Next, we turn to evaluate the estimation accuracy under different number of users. In this part of the experiment, we still set the service demand distribution with known values of the generated model, and vary the number of users in $N \in \{1, 5, 10\}$. The assigned parameters of service demand in the generated models are $m = 0.7$ and $c^2 = 16$, which represents a higher variability than in the first numerical experiment. During solving generated models, we notice that with $N = 10$ the CPU utilization can reach over 95% which is close to saturation. As shown in Figure 7, all the green curves with baseline algorithm fail to model the response time distribution accurately under different workloads. While, the fitting results of our method achieve very close fit to the trace data under light load N. For $N = 5$ and $N = 10$, the results of the proposed method still yields good performance for both body and tail distribution compared to the baseline.



(a) $N = 50, U = 33\%$ (b) $N = 100, U = 43\%$ (c) $N = 300, U = 95\%$

Fig. 8. CCDF of response time with different utilization for the fitted models with FCFS.

Table 2. Service demand distribution parameter estimation results for FCFS

N	$m \times (10^{-4})$	c^2	ν	$Likelihood \times (10^4)$
50	8.33	8.24	11	6.62
100	6.21	11.22	57	14.31
300	3.50	8.03	12	10.64

Table 3. Service demand distribution parameter estimation results for FCFS with FCFS-single method

N	$m \times (10^{-4})$	c^2	ν
50	13.61	2.31	4
100	7.96	2.85	5
300	3.72	2.87	7

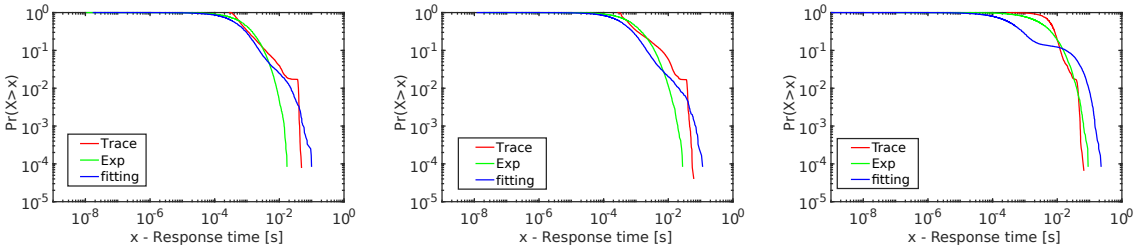
7.4 Analysis of Results on Measured Traces

Analysis with FCFS. We now turn our attention to experiments of the real trace on the microservice application. First, we present our experimental results with FCFS scheduling policy under different workload patterns.

We notice that for a single server queueing system under FCFS scheduling, the service times are measurable by sampling IDTs when the server is not idle, which provides an explicit way to evaluate our proposed estimation method. Therefore, for FCFS scheduling, we add another comparison by directly calculating the first three moments from these sampled service times, which is denoted the *FCFS-single* method. We evaluate the model prediction results with FCFS-single method, baseline algorithm, and our method to the real trace data. The CCDF of response times for low, medium, and high CPU utilization are plotted in Figure 8, respectively. The estimated parameters for service demand distribution with FCFS are summarized in Table 2, and the parameters based on FCFS-single method are in Table 3.

First, compared to the baseline algorithm with exponential distribution, it can be seen that for all 3 different N, our method produces a closer fit. As one can see for the body fitting, both baseline and our proposed model yield good performance. For $N = 50$ and $N = 100$, our model fits the body with better accuracy, whereas the baseline method is outperformed for $N = 300$. However, for the fitting of the tail, the baseline method is worse in terms of accuracy for all values of N.

Then we compare the fitting results of our heuristic method to the direct measurement - FCFS-single method. We observe in Figure 8 that our estimation results yield a similar accuracy of fitting the tail of the distribution under low and medium utilization levels as the FCFS-single method. Both the blue and black curves are much closer to the one of the real trace data. While our method achieves more accurate fitting of the body when $N = 100$. For higher CPU



(a) $N = 50, U = 33\%$ (b) $N = 100, U = 43\%$ (c) $N = 300, U = 95\%$

Fig. 9. CCDF of response time with different utilization for the fitted models with PS.

utilization, the results from our method show slightly weaker fitting in the middle of the body, but achieve a better fit for the tail of the distribution. We interpret that the real microservice system is not precisely served by FCFS scheduling, while it exhibits some state-dependent degree of CPU sharing. The proposed heuristic method captures instances of large service variance that occur in certain system states, which can be inferred from c^2 in Table 3.

Overall, under FCFS scheduling policy, our estimated distribution achieves a better fit throughout the distribution curve, producing an approximation of the real trace distribution accurately. Compared to the baseline and FCFS-single method, our proposed method can model the tail behaviors, which is significant to model different intensive workloads.

Analysis with PS. To analyze the impact of scheduling policies, we conduct experiments with $N \in \{50, 100, 300\}$ and PS scheduling policy. The parameter estimation results with PS scheduling are shown in Table 4 and the fitting comparison is plotted in Figure 9.

First, compared to the results with FCFS scheduling, switching to PS scheduling impacts the parameter estimation results of the baseline in a significant manner. Comparing the estimated skewness to the results for FCFS in Table 2, we can observe smaller values of skewness in Table 4. While the likelihood values for different workloads with PS are still close to the one with FCFS.

For the fitting comparison on CCDF diagrams, under light and medium load, it can be seen in Figure 9(a) and 9(b), the baseline method first fits well at the beginning, and it decays faster from the middle body, ultimately not able to capture the tail for light and medium load. While our method also fits the body with low inaccuracy and outperforms the baseline for the tail fitting, showing a more accurate fit for the distribution. However, with increasing the number of users to $N = 300$ that is close to 100% CPU utilization, the baseline method achieves a closer fit to the data with PS scheduling in Figure 9(c).

The above experimental results with FCFS and PS scheduling indicate that in reality, the actual system scheduling will factor in several elements such as caching, memory bandwidth, and operating system scheduling, which are neither perfectly PS nor FCFS. Therefore, our results indicate that either model provides a reasonable approximation to the observed system behavior, but FCFS appears more suitable to model heavy loads.

Summarizing, the previous results indicate that in the majority of instances the proposed method is able to estimate the service demand distribution with higher fidelity than simple exponential service demand. By comparing the CCDF of the response time distribution, the proposed heuristic method can also capture the tail behaviors with higher accuracy.

8 Heuristic estimation for multiclass models

For a microservice, different services can be identified into the classes based on different endpoints to model heterogeneous service processing. In this section, we generalize our service demand distribution estimation method to

Table 4. Service demand distribution parameter estimation results for PS

N	$m \times (10^{-4})$	c^2	ν	$Likelihood \times (10^4)$
50	8.33	8.24	9	6.11
100	6.21	11.22	10	14.53
300	3.50	8.03	4	10.68

the multiclass service queues. Single-class service demands are often insufficient to capture the dynamic behaviors in the server, resulting in inaccurate predictions. For example, ignoring the presence of multiple classes can systematically underestimate or overestimate the service demand distribution for a queueing station.

Multiclass service demand distribution estimation can, however, be intractably expensive to solve with either the global search or the heuristics-based method mentioned in sections 5 and 6. First, the state space explosion problem could be further worsened by the use of different APH distributions for each class, with the state space roughly growing in size as the product of the corresponding state spaces for models with a single class. Moreover, the optimization problem can be challenging due to the increased number of search parameters, which are now to be differentiated across service classes.

To mitigate the execution complexity of searching for the optimal parameters, class aggregation [5, 16] can be applied to service demand distribution estimation. In particular, multiclass estimation problem can be systematically reduced to an equivalent two-class estimation problem. We also introduce an extension of our heuristics-based method for two classes where the model includes only a *tagged* class and an *aggregated* class, in the sense described later in this section. Finally, we evaluate the accuracy and scalability of the multiclass estimation in Section 9.

8.1 Class aggregation.

First, we illustrate the class aggregation process through the example shown in Figure 10. In this procedure, we tackle the estimation of service demand distribution of each class by solving R two-class submodels, with R being the total number of classes in the original model. For the problem in Figure 10, we need to solve three submodels, so that the service demand distribution for a given job class (e.g., class A) is estimated based on a submodel with a tagged class (e.g., A) and an aggregated class that includes all other classes (e.g., B and C). By recursively applying this approach to all classes, considering each in turn as the tagged one, the problem is mapped to a collection of submodels, which can be solved independently of each other. In essence, this approach approximates the service demand distribution in a more scalable fashion than the original model by reducing the level of fidelity by which we represent the other classes. When solving each submodel, we only extract from that submodel the estimated service demand distribution of the tagged class.

The detailed description of submodel generation based on class aggregation is as follows. Referring to Figure 10, assume our objective is to estimate the service demand distribution for the tagged class A. In the aggregated class, the number of users is directly calculated with $N_{agg} = N_B + N_C$. Let denote p as the probability to be as class B; $1 - p$ is the probability to be class C. The probability for the aggregated class is decided by the throughputs of each class, for example, $p = \frac{X_B}{X_B + X_C}$, where X_i represents the throughput of class i. The aggregated mean value m^A_{agg} for the submodel from tagged class A is calculated with the throughputs and the probabilities of the aggregated classes. Similarly, the aggregated SCV and the skewness can be obtained as a mixture distribution in the same way, where the original parameters of the service demand distribution for each class are the component of the mixture. With this definition, the moments have the following expression

$$E[M^i_{agg}] = \sum_{p \in P, c \in C} pE[M^i_c] \tag{13}$$

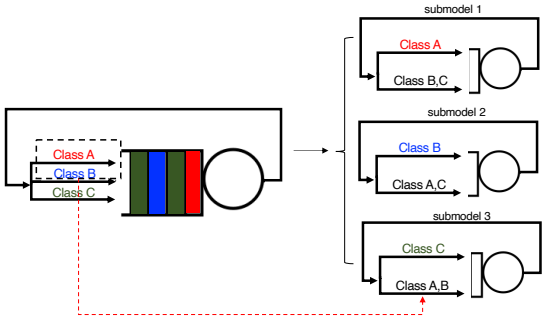


Fig. 10. Example: submodel generation based on class aggregation with 3 classes

where M_c^i represents the i^{th} moment of the service demand distribution of class c and $\boldsymbol{C}$ is the set of service classes. $\boldsymbol{P}$ represents the probabilities of each class except the tagged class.

Multiclass MAP. Multiclass service demand distribution is an extension of the single-class case. Multiclass departure process MAP can also be represented with $MAP = \{\boldsymbol{D}_0, \boldsymbol{D}_1\}$, where $\boldsymbol{D}_1 = \sum_{i=1}^{N} D_1^c$. Let $\boldsymbol{\pi}_c$ be the stationary distribution that follows a departure of class c that satisfies $\boldsymbol{\pi}_c = \pi(-\boldsymbol{D}_0)^{-1}\boldsymbol{D}_1^c$.

For each inter-departure time X_i extracted from the system monitoring traces, we denote c_i as the class label of the current departure job. The value of c_i can be repeated within the class set $\boldsymbol{C}$ of the queueing model, e.g., $\boldsymbol{C} = [c_1, c_2, c_2, c_3, \cdots]$.

For multiclass MAPs, we assume that the IDTs and the class labels of the departure jobs are independent. Thus, the PDF of IDTs is

$$f(\boldsymbol{X}, \boldsymbol{C}) = \boldsymbol{\pi} e^{\boldsymbol{D}_0 X_1} \boldsymbol{D}_1^{c_1} e^{\boldsymbol{D}_0 X_2} \boldsymbol{D}_1^{c_2} \dots e^{\boldsymbol{D}_0 X_n} \boldsymbol{D}_1^{c_n} \boldsymbol{e} \tag{14}$$

Based on (14), the log joint PDF for multiclass can be approximated with

$$\log f(\boldsymbol{C}, \boldsymbol{X}) = \sum_{i=1}^{K} \log(\boldsymbol{\pi} e^{\boldsymbol{D}_0 X_i} \boldsymbol{D}_1^{c_i} \boldsymbol{e}) \tag{15}$$

Multiclass MEAN and SCV estimation. For multiclass service demand distribution estimation, we also aggregate classes for the third-moment estimation. The mean value of service demand is calculated referring to the response-time based estimation extended to the multiclass scenario. Let

- Response times of class c from the monitoring traces be represented as $\boldsymbol{R}_c$.
- Queue length of class j seen upon arrival of class c be A_c^j.

For multiclass queue, the two scheduling policies we consider have different mean estimation expressions due to different behaviors of the server. For FCFS, we need to consider the different classes of jobs queueing in the server to wait for the service

$$E[D_c] = \frac{E[R^c] - \sum_{i \in C, i \neq c} E[D_i] A_c^i}{1 + A_c^c} \tag{16}$$

While for PS, the mean service demand can be estimated with Equation (17) which is directly extended from the single class.

$$E[D_c] = \frac{E[R^c]}{1 + \sum_{i \in C} A_c^i} \tag{17}$$

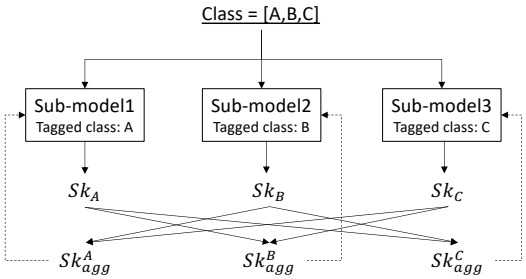


Fig. 11. Synchronize submodels with iterations

SCV estimation c^2 for class c can be generalized with multiclass jobs using the following equation

$$c_c^2 = \max \frac{E[(R_{ki}^{'c} - E[R_k^{'c}])^2]}{E[R_k^{'c}]^2} \tag{18}$$

For each class c, we group the response times using the same method as for the single class case, where $R_k^{'c}$ is the grouped set of response time with queue length A_k.

Iterative synchronization of the submodels. To synchronize the submodel for the tagged class with the estimated moments by the heuristic estimation method, we then iterate on the results of solving submodels. As mentioned before, the service demand distribution for R different classes is estimated considering a different tagged class each time. However, also the service demand distribution results for the aggregation classes can be used to calibrate the accuracy of the estimation iteratively, as shown in Figure 11.

We define an iterative method to solve all $|C|$ two-class submodels with the $|C|$ groups of estimated parameters with m, c^2 and ν for each class. Here, we denote ν_{agg}^c as the skewness value for the aggregated class in a submodel with a tagged class c. The parameters of each aggregated class (e.g., ν_{agg}^A) can be updated after obtaining the skewness values (e.g., calculate a new $\nu_{agg}^{A\,'}$ with ν_B and ν_C using Equation (13)) from the current estimation iteration. Thus, the updated parameters of the aggregated classes in each iteration can be used to synchronize the submodel, which provides a calibration on the moments for aggregated classes. With iterative calibration of fitting the APH distribution of service demand in the queue node for aggregation class, the estimated skewness would be converged.

8.2 Heuristic based estimation method for multiclass demand

Service demand distribution estimation for multiclass can be split into two sub-problems. The first one is to estimate skewness for each class with the tagged class and aggregated class. The second problem is to iteratively calibrate the accuracy of the estimated skewness.

Assuming that there are $|C|$ job classes in total, we need to generate $|C|$ submodels to solve this problem. In each submodel, there are two classes of jobs with N_{tag} and N_{agg} users. The heuristics-based method for multiclass is implemented as shown in Algorithm 3. This algorithm requires the following inputs. The throughput $\boldsymbol{X}$ for different classes (calculated from monitoring traces), the number of users N_c and think time Z_c in each class, and the clustered set of IDTs and class labels.

We define the function *Likelihood_multi* to calculate the likelihood on multi-class departure MAPs in Algorithm 3 referring to Line 3-25. In this procedure, the service demand distribution is first fitted conditionally on the synchronized iteration with the prior information of the skewness as shown from Line 7 to 11. Then a two-class submodel for estimation current ν_c is generated. The submodel is then solved by analyzing the CTMC to obtain the infinitesimal

Algorithm 3 Heuristics based estimation method for multiclass

Input: $C \leftarrow$ service class set, $\boldsymbol{m}, \boldsymbol{c}^2 \leftarrow$ mean and SCV set, $\boldsymbol{Z} \leftarrow$ user think times, $\boldsymbol{X} \leftarrow$ throughputs for different service classes, $\boldsymbol{Y} = (\boldsymbol{Y_1} = [Y_{11}, \cdots, Y_{1n}, l_{11}, \cdots, l_{1n}], \boldsymbol{Y_2} = [Y_{21}, \cdots, Y_{2n}, l_{21}, \cdots, l_{2n}]) \leftarrow$ Set of clustered inter-departure times for each service class, $\boldsymbol{N} \leftarrow$ Set of number of users in each class

1: $\boldsymbol{C}, \boldsymbol{\mu_3} \leftarrow CHECKScv(\boldsymbol{C}, \boldsymbol{m}, \boldsymbol{c}^2, \boldsymbol{Z})$ /*Defined in Algorithm 4*/
2: Initialize $iter = 1$
3: **function** Likelihood_multi($\mu_1, \mu_2, \mu_3, iter, \boldsymbol{Y}$)
4: $APH_1(\boldsymbol{\alpha}, \boldsymbol{T}) \leftarrow APHFit[\mu_1, \mu_2, \mu_3]$
5: $m_{agg}, c^2_{agg}, \nu_{agg} \leftarrow m, c^2, \nu, \boldsymbol{X}$ /*From Equation (13)*/
6: Compute the first three moments for aggregated class $[\mu_1^{'}, \mu_2^{'}, \mu_3^{'}]$ based on Equations (1) to (3)
7: **if** $iter == 1$
8: $APH_2(\boldsymbol{\alpha}^{'}, \boldsymbol{T}^{'}) \leftarrow APHFit[\mu_1^{'}, \mu_2^{'}]$
9: **else**
10: $APH_2(\boldsymbol{\alpha}^{'}, \boldsymbol{T}^{'}) \leftarrow APHFit[\mu_1^{'}, \mu_2^{'}, \mu_3^{'}]$
11: **end if**
12: **if** APH_1 and APH_2 are feasible
13: $submodel \leftarrow GENERATEModel(APH_1, APH_2, N_{tag}, N_{agg})$
14: $\boldsymbol{Q} \leftarrow solve(submodel)$
15: $MAP \leftarrow \{D_0, D_1^{tag} + D_1^{agg}\}$, generate π
16: **for** $k = 1$ to n
17: $\boldsymbol{\beta}_1 \leftarrow CTMCUniform(\pi, D_0, Y_{1k})$
18: $\boldsymbol{\beta}_2 \leftarrow CTMCUniform(\pi, D_0, Y_{2k})$
19: $L \leftarrow L + l_{1k} \log(\beta_1 D_1^{tag} \boldsymbol{e}) + l_{2k} \log(\beta_2 D_1^{agg} \boldsymbol{e})$
20: **end for**
21: **else**
22: $L \leftarrow -\infty$
23: **end if**
24: **return** L
25: **end function**
26: **while** $tol \neq 0$ **do**
27: **for** i in C **do**
28: Relabel samples with tagged and aggregated classes
29: $\boldsymbol{Qlen} \leftarrow reCalculate(\boldsymbol{Qlen})$
30: Update users with two classes $\boldsymbol{N_{tag}}, \boldsymbol{N_{agg}}$
31: $\boldsymbol{\mu_3} \leftarrow m, c^2, \boldsymbol{\nu}$ with Equation (3)
32: $\mu_3^* = \underset{\boldsymbol{\mu_3}}{\operatorname{argmax}}\, Likelihood\_multi(\mu_1, \mu_2, \mu_3, iter, \boldsymbol{Y})$
33: **end for**
34: $[\mu_1^{'}, \mu_2^{'}, \mu_3^{'}] \leftarrow momentAgg(\boldsymbol{X}, [\mu_1, \mu_2, \mu_3^*])$
35: $\nu_{agg}^{'} = \nu_{agg}$
36: $\nu_{agg} \leftarrow \mu_3^{'}$
37: $tol = diff(\nu_{agg}, \nu_{agg}^{'})$
38: $iter = iter + 1$
39: **end while**

generator Q. The two-class departure MAP now can be constructed as the process $(D_0, D_1^{tag}, D_1^{agg})$, which are generated by filtering different classes on Q. In Line 11 to 15, the result of log-likelihood for the tagged class with relabeled inter-departure times is obtained.

Algorithm 4 *CHECKScv*

Input: $C \leftarrow$ service class set, $\boldsymbol{m}, \boldsymbol{c}^2 \leftarrow$ mean and SCV set, $\boldsymbol{Z} \leftarrow$ user think times

1: $\boldsymbol{m}_{agg} \leftarrow meanAgg(\boldsymbol{Z}, \boldsymbol{m})$ /*According to Equation (13)*/
2: $\boldsymbol{c}^2_{agg} \leftarrow scvAgg(\boldsymbol{Z}, \boldsymbol{c}^2)$ /*Using Equation (13) and Equations (1) to (2)*/
3: **for** $i = 1$ in $|C|$ **do**
4: **if** $c_i^2 \leq 1$ **then**
5: **if** $c_i^2 == 1$ **then**
6: $\nu_i = 2$ /*the skewness of Exponential distribution*/
7: **end if**
8: Calculate ν_i of $Erlang(m_i, c_i^2)$
9: $C \leftarrow$ remove i from C
10: Compute μ_{3i} with (m_i, c_i^2, ν_i) using Equation (3)
11: **end if**
12: **end for**
13: **return** $C, \boldsymbol{\mu}_3$

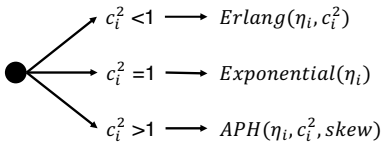
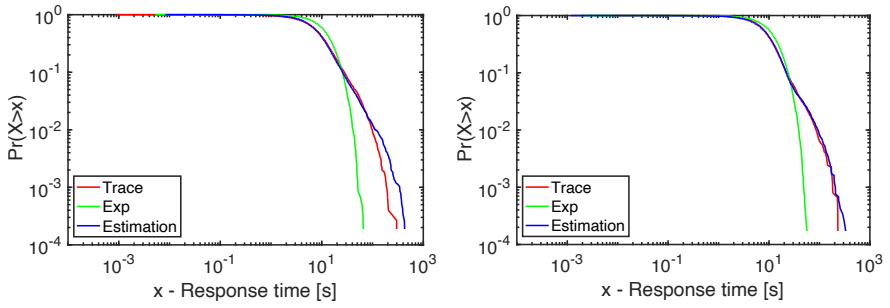


Fig. 12. Conditional skewness estimation

Algorithm 3 starts with calculating the number of classes and the set of third moments according to Algorithm 4. Then it iterates on each service class, updating the number of uses in the tagged and aggregated class and relabel the samples with two classes, referring to Line 28-30. The algorithm calculates the third moment of the current tagged service class and obtain the likelihood value via *Likelihood_multi*. Then the first three moments of the current tagged class can be then obtained with the third moment candidate that maximized the likelihood. The parameters of the skewness of the remaining classes are solved with the same procedure by solving with submodels as illustrated.

After solving all submodels based on the MLE method, the skewness of each class can be obtained. The submodels can be synchronized with updated aggregated skewness values. The aggregated skewness for each submodel is calibrated with moment aggregation based on the throughputs. At the end of each iteration, the algorithm compares the current skewness for the aggregated classes in each submodel with the updated values.

To accelerate the convergence of the algorithm, we add additional conditions to estimate the skewness as shown in Figure 12 and Algorithm 4.For each service class, there are three cases to fit for the APH distributed service demand in the queue node. First, if $c_i^2 < 1$, it can be then fitted from an Erlang distribution for the service demand with m_i, c_i^2. Thus, there is no estimation of the skewness for class i since it can be directly calculated based on the Erlang distribution. In this case, Algorithm 3 only involves the rest of class in C except class i according to Algorithm 4. Next, if $c_i^2 = 1$, the service demand distribution of this class should form an Exponential distribution with the skewness equal to 2.



(a) 3 classes (b) 4 classes

Fig. 13. CCDF of response times of multiclass for numerical experiments

Table 5. Parameter estimation results for numerical experiments with 3 classes

	m	*Estimated* m	c^2	*Estimated* c^2	ν	*Estimated* ν
Class1	5	5.15	1	1.62	2.00	50.0
Class2	5	5.13	4	4.36	5.31	5.4
Class3	5	5.00	8	18.96	7.98	12.8

9 Multiclass experiments

9.1 Numerical experiments

To evaluate the effectiveness of the proposed heuristic-based estimation for multiclass jobs, numerical experiments are conducted based on traces collected via simulation.

In the following experiments, we first generate closed queueing models with 3 service classes of service, and sample from the simulation results to construct the datasets for evaluation. To avoid the difference in service distributions being driven by their mean value, we set the same mean value for the service demand distribution for different classes in the queue node to $m = 5$ and c^2 is selected as $1, 4, 8$ for each class. In this way, we can have the ground truth for comparison to the result of our proposed method. The details of the parameter settings in the sampling models are shown in Table 5 and Table 6. In the experiment of Table 5, we set the mean and SCV values for the generated model. Thus, ν of 3 classes can be compared to the baseline (Column ν) by calculating the skewness of the fitted APH with m and c^2. While for Table 6, we also set the skewness values in $\nu = [4, 7, 10]$ of the generated model. Therefore, in the second experiment, the accuracy of the skewness estimation can be directly measured by comparing the setting skewness and the experimental results. The CCDF of the above experiments are shown in Figure 15(a) and 15(b).

First, compared to the baseline algorithm with exponential distribution, our estimation results yield better performance from the body to the tail. The green curve of the exponential distribution decays faster from the middle, resulting in an inaccurate prediction of the tail of the response times. We can then observe from the parameter estimation results in Table 5 that the estimated values of m and c^2 are close to the given parameters of the generated model. If the optimal result reaches a bound value, the algorithm will then fit from the first two moments instead of three moments to reduce the inaccuracy. The results in Table 6 show that the skewness estimation for all 3 classes is accurate.

Next, we evaluate the effectiveness of iterative calibration with different submodels. We compare the CCDFs with and without iterative calibration (e.g., with a single execution) in Figure 14. The black curve represents the CCDF without iteration. It can be seen that for the body of the distribution, there is no obvious difference between our estimation and the one without calibration. However, our estimation method captures the tail of the distribution quite well compared

Table 6. Parameter estimation results for numerical experiments with 4 classes

	m	*Estimated* m	c^2	*Estimated* c^2	ν	*Estimated* ν
Class1	5	5.06	1	1.55	4	8.1
Class2	5	4.40	4	4.14	7	8.4
Class3	5	4.89	8	10.77	10	12.3

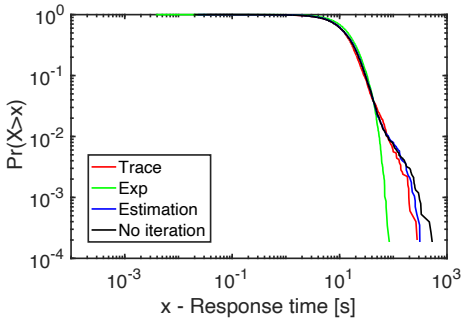


Fig. 14. Comparison results for iterative calibration

to the simulated trace data. Instead, the method without iteration deviates from the tail. Therefore, we conclude that, for the multiclass case, the calibration with iteration on the skewness can help to improve the accuracy of the service demand distribution estimation for multiclass cases.

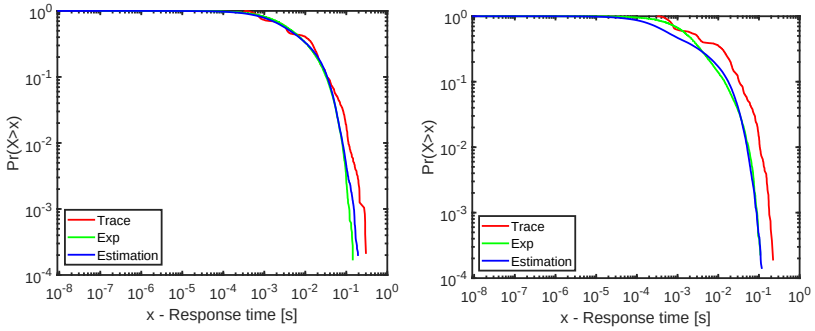
9.2 Real traces analysis

In the following experiments, we evaluate our estimation method with real traces from the same microservice application (Sock Shop) as before. We concluded from Section 7 that the scheduling policy under high load is intended to be FCFS, while more likely to be PS with lower load. In this section we have run experiments with two parameterizations: $N = \{10, 10, 10\}$ and $N = \{10, 10, 10, 10\}$ for 3 and 4 service classes, which are under medium load, configuring the worst case for estimation, since neither FCFS nor PS will constitute a perfect fit.

Analysis with FCFS. Comparative results for $N = 30$ for 3 classes and $N = 40$ with 4 classes are shown in Figure 15(a) and Figure 15(b) respectively. Table 7 shows the corresponding parameter estimation results for the service demand distribution. Note that c^2 of class 3 is less than one, and the service demand distribution for class 3 is estimated with an Erlang distribution as illustrated in Section 8. Thus, in these two tables, ν of class 3 is obtained by calculating the skewness of the Erlang distribution.

For three classes, both the baseline algorithm and our estimation are able to fit the response time distribution of the real trace, while our estimation is better on the tail. It can be seen from Figure 15(b) that our estimation achieves almost the same fitting accuracy as the exponential one. Compared to the baseline algorithm with FCFS for both three and four classes, the heuristics-based method performs similar as the exponential distribution, however, the fitting from our estimation is still a fair approximation of the real trace behavior.

Analysis with PS. Table 8 and Figure 16 show the estimation results with PS. For fitting the service demand distribution of three service classes in Figure 16(a), both our estimation result (the blue curve) and the exponential one (green) are close to the real trace, indicating a good characterization of the service demand of the real system. Figure 16(b) shows how our estimation method outperforms the baseline algorithm for the four classes case. Moreover, comparing the fitting on the tail of the distribution, our result is closer to the real trace. Instead, with the exponentially distributed



(a) 3 classes (b) 4 classes

Fig. 15. CCDF of response times for multiclass with FCFS

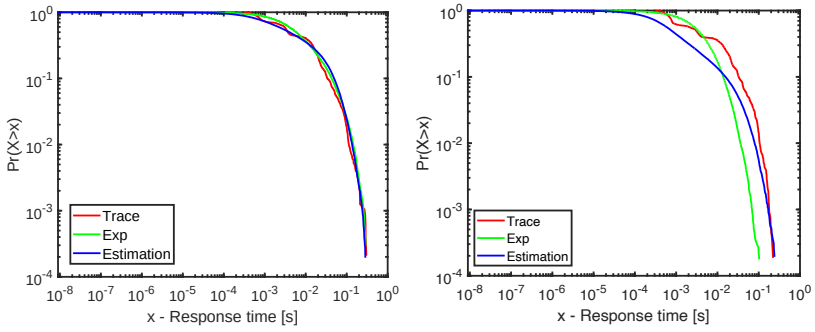
Table 7. Parameter estimation results with FCFS

3 classes

	$m \times 10^{-3}$	c^2	ν
Class 1	1.8	11.1	16.1
Class 2	4.4	2.4	14.9
Class 3	17.5	0.7	1.6

4 classes

	$m \times 10^{-3}$	c^2	ν
Class1	1.4	10.2	14.3
Class2	3.8	2.5	12.9
Class3	16.1	0.7	1.6
Class4	1.4	10.0	15.1



(a) 3 classes (b) 4 classes

Fig. 16. CCDF of response times for multiclass with PS

service demand, the predictive curve of the response times decays faster, especially for the four classes trace. Thus, the heuristics-based method has a better fit throughout the response time distribution.

In summary, for three and four service classes the estimation results with both FCFS and PS achieve reasonable accuracy to capture the real system. We can draw the same conclusion as the single-class experiments that the tail behaviors based on our estimation achieve good accuracy to capture the response times distribution, especially for high percentiles, while also being more efficient to compute.

10 Conclusion

In this paper, we introduced a service demand distribution estimation method by using Markovian arrival processes to abstract a microservice. We presented a closed queueing model for a microservice and characterized the service demand

Table 8. Parameter estimation results with PS

3 classes

	$m \times 10^{-3}$	c^2	ν
Class 1	2.8	11.1	5.1
Class 2	5.7	2.4	3.9
Class 3	26.1	0.7	1.6

4 classes

	$m \times 10^{-3}$	c^2	ν
Class1	3.2	10.2	5.5
Class2	5.7	2.5	3.5
Class3	22.8	0.7	1.6
Class4	3.0	10.0	5.4

of the queue node with an APH distribution based on departure MAP. For multi-class service demand distribution estimation, we applied class aggregation to reduce the number of classes, mapping to a collection of two-class service demand estimations, and introducing an iterative calibration to increase the accuracy of the two-class estimation by combining their individual results. Heuristics-based estimation method was proposed to reduce the computational cost compared to the global search approach. We further showed that the proposed estimation method with both optimization and heuristic solution yields a better performance on fitting real traces of microservices compared to state-of-the-art alternatives.

As part of the future work, we plan to integrate our work in an orchestration toolchain and extend it to microservices architectures and serverless function chains with LQNs. In such cases, a single component of a microservice or an individual serverless function can still be abstracted as a single queueing model, suggesting that the currently proposed estimation method can possibly capture the service demand distribution based on data from distributed tracing and monitoring.

References

[1] [n. d.]. ([n. d.]).

[2] Allan T Andersen, Marcel F Neuts, and Bo Friis Nielsen. 2000. Lower order moments of inter-transition times in the stationary QBD process. *Methodology and Computing in Applied Probability* 2, 4 (2000), 339–357.

[3] Jonatha Anselmi, Bernardo D'Auria, and Neil Walton. 2013. Closed queueing networks under congestion: Nonbottleneck independence and bottleneck convergence. *Mathematics of operations research* 38, 3 (2013), 469–491.

[4] Maury Bramson, Yi Lu, and Balaji Prabhakar. 2010. Randomized load balancing with general service time distributions. *ACM SIGMETRICS performance evaluation review* 38, 1 (2010), 275–286.

[5] Alexandre Brandwajn and Thomas Begin. 2019. First-come-first-served queues with multiple servers and customer classes. *Performance Evaluation* 130 (2019), 51–63.

[6] Peter Buchholz, Peter Kemper, and Jan Kriege. 2010. Multi-class Markovian arrival processes and their parameter fitting. *Performance Evaluation* 67, 11 (2010), 1092–1106.

[7] Giuliano Casale. 2011. Building accurate workload models using markovian arrival processes. *ACM SIGMETRICS Performance Evaluation Review* 39, 1 (2011), 357–358.

[8] Giuliano Casale, Paolo Cremonesi, and Roberto Turrin. 2007. How to select significant workloads in performance models. In *CMG Conference Proceedings*. Citeseer, 58–108.

[9] Giuliano Casale, Paolo Cremonesi, and Roberto Turrin. 2008. Robust workload estimation in queueing network performance models. In *16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008)*. IEEE, 183–187.

[10] Srinivas R Chakravarthy. 2019. Busy Period Analysis of GI/G/c and MAP/G/c Queues. In *Performance Prediction and Analytics of Fuzzy, Reliability and Queuing Models*. Springer, 1–31.

[11] Paolo Cremonesi, Paul J Schweitzer, and Giuseppe Serazzi. 2002. A unifying framework for the approximate solution of closed multiclass queuing networks. *IEEE Trans. Comput.* 51, 12 (2002), 1423–1434.

[12] Sergei Dudin and Olga Dudina. 2019. Retrial multi-server queuing system with PHF service time distribution as a model of a channel with unreliable transmission of information. *Applied Mathematical Modelling* 65 (2019), 676–695.

[13] Martin Duggan, Karl Mason, Jim Duggan, Enda Howley, and Enda Barrett. 2017. Predicting host CPU utilization in cloud computing using recurrent neural networks. In *2017 12th International Conference for Internet Technology and Secured Transactions (ICITST)*. IEEE, 67–72.

[14] Greg Franks, Tariq Al-Omari, Murray Woodside, Olivia Das, and Salem Derisavi. 2008. Enhanced modeling and solution of layered queueing networks. *IEEE Transactions on Software Engineering* 35, 2 (2008), 148–161.

[15] Michele Garetto, R Lo Cigno, Michela Meo, and M Ajmone Marsan. 2001. A detailed and accurate closed queueing network model of many interacting TCP flows. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE*

Computer and Communications Society (Cat. No. 01CH37213), Vol. 3. IEEE, 1706–1715.

[16] Balbo Gianfranco, Bruell Steven C, and Ghanta Subbarao. 1986. The solution of homogeneous queueing networks with many job classes. *Journal of Systems and Software* 6, 1 (1986), 41–53.

[17] Elena Grigoreva, Maximilian Laurer, Mikhail Vilgelm, Thomas Gehrsitz, and Wolfgang Kellerer. 2017. Coupled markovian arrival process for automotive machine type communication traffic modeling. In *2017 IEEE International Conference on Communications (ICC)*. IEEE, 1–6.

[18] András Horváth and Miklós Telek. 2002. Phfit: A general phase-type fitting tool. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*. Springer, 82–91.

[19] Bara Kim, Jeongsim Kim, and Jerim Kim. 2010. Tail asymptotics for the queue size distribution in the MAP/G/1 retrial queue. *Queueing Systems* 66, 1 (2010), 79–94.

[20] Alexander Klemm, Christoph Lindemann, and Marco Lohmann. 2003. Modeling IP traffic using the batch Markovian arrival process. *Performance Evaluation* 54, 2 (2003), 149–173.

[21] K Krishna and M Narasimha Murty. 1999. Genetic K-means algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 29, 3 (1999), 433–439.

[22] Achyutha Krishnamoorthy, R Manikandan, and B Lakshmy. 2015. A revisit to queueing-inventory system with positive service time. *Annals of Operations Research* 233, 1 (2015), 221–236.

[23] Shasha Liao, Hongjie Zhang, Guansheng Shu, and Jing Li. 2017. Adaptive resource prediction in the cloud using linear stacking model. In *2017 Fifth International Conference on Advanced Cloud and Big Data (CBD)*. IEEE, 33–38.

[24] Paulo Maciel, Rubens Matos, Bruno Silva, Jair Figueiredo, Danilo Oliveira, Iure Fé, Ronierison Maciel, and Jamilson Dantas. 2017. Mercury: Performance and Dependability Evaluation of Systems with Exponential, Expolynomial, and General Distributions. In *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*. 50–57.

[25] M Ajmone Marsan, Gianfranco Balbo, Gianni Conte, Susanna Donatelli, and Giuliana Franceschinis. 1998. Modelling with generalized stochastic Petri nets. *ACM SIGMETRICS performance evaluation review* 26, 2 (1998), 2.

[26] Manar Mazkatli and Anne Koziolek. 2018. Continuous integration of performance model. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*. 153–158.

[27] Cleve Moler and Charles Van Loan. 2003. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM review* 45, 1 (2003), 3–49.

[28] Marcel F Neuts. 1992. Models based on the Markovian arrival process. *IEICE Transactions on Communications* 75, 12 (1992), 1255–1265.

[29] Marcel F Neuts and Kathleen S Meier. 1981. On the use of phase type distributions in reliability modelling of systems with two components. *Operations-Research-Spektrum* 2, 4 (1981), 227–234.

[30] Hiroyuki Okamura and Tadashi Dohi. 2009. Faster maximum likelihood estimation algorithms for Markovian arrival processes. In *2009 Sixth international conference on the quantitative evaluation of systems*. IEEE, 73–82.

[31] Takayuki Osogami and Mor Harchol-Balter. 2003. A closed-form solution for mapping general distributions to minimal PH distributions. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*. Springer, 200–217.

[32] Colm Art O'Cinneide. 1990. Characterization of phase-type distributions. *Stochastic Models* 6, 1 (1990), 1–57.

[33] Jian-Xin Pan and Kai-Tai Fang. 2002. Maximum likelihood estimation. In *Growth curve models and statistical diagnostics*. Springer, 77–158.

[34] A Parini and Achille Pattavina. 2005. Modelling voice call interarrival and holding time distributions in mobile networks. In *Proc. of ITC 2005*. 729–738.

[35] Juan F Pérez and Giuliano Casale. 2017. Line: Evaluating software applications in unreliable environments. *IEEE Transactions on Reliability* 66, 3 (2017), 837–853.

[36] JuaHon F Pérez, Sergio Pacheco-Sanchez, and Giuliano Casale. 2013. An offline demand estimation method for multi-threaded applications. In *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE, 21–30.

[37] Andrew Reibman and Kishor Trivedi. 1988. Numerical transient analysis of Markov models. *Computers & Operations Research* 15, 1 (1988), 19–36.

[38] Jerome Rolia and Vidar Vetland. 1998. Correlating resource demand information with arm data for application services. In *Proceedings of the 1st international workshop on Software and performance*. 219–230.

[39] Roger B Sidje, Kevin Burrage, and Shev MacNamara. 2007. Inexact rouniformization method for computing transient distributions of Markov chains. *SIAM Journal on Scientific Computing* 29, 6 (2007), 2562–2580.

[40] Simon Spinner, Giuliano Casale, Fabian Brosig, and Samuel Kounev. 2015. Evaluating approaches to resource demand estimation. *Performance Evaluation* 92 (2015), 51–71.

[41] Axel Thummler, Peter Buchholz, and Miklós Telek. 2006. A novel approach for phase-type fitting with the EM algorithm. *IEEE Transactions on dependable and secure computing* 3, 3 (2006), 245–258.

[42] Runan Wang, Giuliano Casale, and Antonio Filieri. 2021. Service Demand Distribution Estimation for Microservices Using Markovian Arrival Processes. In *International Conference on Quantitative Evaluation of Systems*. Springer, 310–328.

[43] Wei Wang, Xiang Huang, Xiulei Qin, Wenbo Zhang, Jun Wei, and Hua Zhong. 2012. Application-level cpu consumption estimation: Towards performance isolation of multi-tenancy web applications. In *2012 IEEE Fifth International Conference on Cloud Computing*. IEEE, 439–446.