The Journal of Supercomputing (2022) 78:18825–18855
https://doi.org/10.1007/s11227-022-04597-y

Workload characterization and synthesis for cloud using generative stochastic processes

Korrapati Sindhu[1] · Karthick Seshadri[1] · Chidambaran Kollengode[2]

Accepted: 8 May 2022 / Published online: 13 June 2022

Abstract

In the recent past, we are witnessing a proliferation in the number of web/mobile applications being hosted on a service provider's Cloud. This has led to a surge in the traffic to the data centers hosting Virtual Machines (VM) running the cloud instances. In a cloud environment, a workload is defined as the requests coming in for the applications which are hosted on VM instances. Workload characterization helps in modeling the associations and correlations in the workload. Workload characterization models that are representative of the ground truth, can be leveraged for: (i) an accurate capacity planning, (ii) better resource utilization, (iii) reducing the spin-up times of VM instances, and (iv) maintaining compliance with Service Level Agreement (SLA). We propose a first-of-its-kind generative Dirichlet process-based model using Latent Dirichlet Allocation (LDA) for workload characterization. The characterization model is dependency preserving, regularized, and generative in nature, that relates the workload to the underlying application or user's behavior that might have generated the workload. To evaluate the descriptive and predictive accuracies of the proposed model, we designed experiments using the Bit Brains Trace (BBT) and Alibaba Cluster Trace. The descriptive accuracy of the proposed workload characterization model is assessed by comparing a synthetic workload against the real workload using Pearson Correlation Coefficient (PCC) and Akaike Information Criterion (AIC) as the metrics. We have also performed statistical tests to assess the similarity between real workload and synthetic workload.

Keywords Cloud computing · Workload characterization · Virtual machines · Dirichlet process · Latent Dirichlet allocation · Gibbs sampling · Resource requirement prediction · Synthetic workload generation

✉ Karthick Seshadri
karthick.seshadri@nitandhra.ac.in

[1] Department of Computer Science and Engineering, National Institute of Technology Andhra Pradesh, Tadepalligudem, Andhra Pradesh, India

[2] Data and AI Platforms, LinkedIn, Bangalore, Karnataka, India

1 Introduction

Many organizations have commenced migration of their applications to the cloud, due to its ability for on-demand provisioning of resources. Besides on-demand provisioning, other prominent merits of cloud computing include scalability, elasticity, reliability, minimizing energy consumption, efficient task scheduling, resource management, and security. Cloud provides services to the users predominantly using virtualization. Virtualization allows to run multiple virtual machines on a physical machine that helps in efficient resource utilization. However, predicting the number of virtual machines to be provisioned for an application is difficult because of the typical dynamic nature of users interacting with the application.

Workload characterization is a key enabler in realizing truly elastic cloud computing due to its applicability in the tasks like workload scheduling, workload prediction, resource planning, synthetic workload generation, and identifying the kinds of workload that lead to failure conditions. Modeling and analysis of the property of workloads are essential to ensure a balanced mix of jobs across VMs with respect to resource utilization and load. Similarly, for predicting the future workloads and for subsequent resource buffering or planning, an understanding of the patterns exhibited by workloads across different workload categories is important.

Workload characterization is the process of modeling resource demands imposed by workloads to applications hosted on the cloud with respect to key aspects such as amount of CPU requirement, amount of memory required, number of I/O operations involved, and network bandwidth consumed. In this paper, a workload is taken to be the input or requests received by an application hosted on a VM and the resources used for processing the requests in the cloud. A thorough analysis of workloads helps the data center administrators in estimating the type and number of resources required to process a request. This estimation will subsequently help in the efficient mapping or arbitration of requests to Virtual Machines (VM).

Further, an accurate model for workloads can be applied for (i) capacity planning, (ii) task scheduling which accounts for efficient resource utilization, (iii) minimum energy consumption, and (iv) reduction of carbon emissions from data centers. Workload characterization also plays an important role in meeting the Service Level Agreements (SLA) and Quality of Service (QoS) requirements. Any workload characterization model should be capable of encoding the properties of workloads. Workload characterization can be performed either by analyzing the workload traces using statistical measures or by learning a model which can perform characterization.

Most of the existing works [1–4] perform workload characterization either by analyzing statistical properties of incoming requests to a data center or by using clustering techniques like k-Means and these models are developed for a specific type of application. The requests arriving for applications at the data center may be heterogeneous. Heterogeneity is defined in terms of the types of resources required for processing the incoming request to an application. Existing research attempts fail to accurately handle heterogeneity in workloads. Hence, there is a justifiable need for developing a model which is capable of handling heterogeneity in the workloads submitted to a data center. However, developing a model that can characterize the workloads arriving at a data center is a challenging task because of the typical temporal variations in workloads.

Since cloud service providers cannot provide access to workload traces due to confidentiality and security related reasons, there is a justified need to devise stochastic processes to reverse engineer the latent properties of task domains and users from the observable workload properties and for generating synthetic workloads in domains where getting hold of real workloads for analysis is not feasible.

In this paper, we propose an unsupervised model for characterizing workloads in a data center, based on a generative probabilistic graphical model called Latent Dirichlet Allocation (LDA) [5]. The proposed model takes a snapshot of jobs running in a data center and performs characterization. The model can be easily generalized into a time-series model when such a characterization is done across multiple time instants with a common set of applications and tasks across the snapshots.

In the proposed model, we identify the task domain that generates the workload where the workload is modeled as a distribution of task domains, and the task domain is modeled as a distribution over the features that describe the workload. This kind of modeling effectively handles heterogeneity in the type of resources required for processing the workload, as workload is modeled as a distribution over task domains. Another significant advantage of the proposed model is that there is no mandatory requirement by the model to learn from a fixed attribute set. The characterization models developed by using the algorithms like k-Means require each workload to have same set of workload features to describe the workload. Following are the key contributions of this paper:

(i) We propose a first-of-its-kind unsupervised model to characterize workloads in a cloud, that accurately models heterogeneity with respect to resource demands.
(ii) This is a novel attempt that proposes to adopt a probabilistic graphical model based on the Dirichlet stochastic process to characterize cloud workloads.
(iii) The proposed workload characterization model is empirically observed to perform well with respect to both descriptive and predictive abilities.
(iv) We have established the efficiency of the proposed model by generating synthetic workload and AIC criterion is used to quantify the representativeness between the synthetic workload and the real workload.

The rest of this paper has the following layout: Sect. 2 discusses the existing methods for workload characterization. In Sect. 3, we explain in detail the proposed method for workload characterization. In the subsequent section, we present the experimental results obtained for the two workload traces BBT and Alibaba Cluster Trace, and inferences. In the last section, we summarize the paper with future research pointers to extend this thread of research.

2 Related work

The efficiency of a data center depends on the execution of user requests while balancing the load on VMs. Shruti et al. [6] showed the impact of workload characterization in the context of capacity planning and performance management. In this work, the authors performed workload characterization based on behavioral patterns observed in workload traces. Different kinds of patterns observed are namely periodicity patterns, threshold patterns, relationship patterns, and variability patterns. Auto Correlation Function is used to assess the resource utilization (CPU, memory, and network) in each Virtual Machine, and it was observed that behavioral patterns help in capacity planning and identifying SLA violations.

Hani [7] et al. proposed a method for virtual machine characterization by mining hypervisor traces to avoid internal access to each VM. A two-stage clustering is performed for coarse-grained and fine-grained workload characterization. Initially, features are extracted from the hypervisor trace using Trace Compass open-source tool. After extracting the features, a two-stage clustering is performed, where the VMs are grouped into a set of clusters using k-Means. In the second stage, the clusters obtained in the first stage are further partitioned to achieve a fine-grained VM characterization.

Workloads can be classified into different categories based on resource requirements, computing environment, and type of application. Based on the resource requirement, workloads are classified as CPU workload, I/O workload, memory workload, and database workload [8]. Based on the applications, workloads are classified as web workload, social network workload, and video service workload. The web workload consists of HTTP requests from clients, document types, transfer size, distinct requests, time between the successive requests for the same file, and file size distribution [9].

Arijit Khan et al. [10] proposed a method for workload characterization and future workload prediction. In this work, the authors performed workload characterization at group level instead of a VM-level characterization. While analyzing the discretized time series data, it is observed that when applications are executed collaboratively by a set of VMs, then there is a change in workload in a correlated manner. VMs are grouped by using the co-clustering technique. Experiments are conducted on 21 days of CPU utilization data collected from an enterprise customer.

Menasce et al. [11] proposed a method for characterizing workloads of an e-commerce site. A Customer Behavior Model Graph (CBMG) is constructed for each session; a session is a sequence of requests by the same customer. The nodes represent requests and edges represent the transition probabilities between states. Once the CBMG is constructed, the next task is determining the parameters that are used to characterize the workload. Parameters that characterize the workload are classified into two categories namely, workload intensity parameters and resource usage parameters. Workload intensity parameters include session arrival rate and average think time between the requests. Resource usage parameters describe the resources used for processing the requests at server side.

In the literature [1–3], several statistics namely min, max, mean, covariance, order statistics, and standard deviation are used for workload characterization. Aragon et al. [12], performed analysis and characterization of workloads of a web application which is implemented using microservices architecture. Different workload features that are considered for characterization are microservice popularity, variability in request intensity, burstiness in request arrivals, inter-arrival time and service time of a request. After analyzing the workload features, a workload can be generated by identifying the statistical distribution characterizing the features.

Shekhawat et al. [13] proposed a method for characterizing workloads based on resource usage. For workload classification and characterization, two datasets namely, Google Cluster Trace (GCT) and Bit Brains Trace (BBT) are used. The authors used six different machine learning algorithms namely Support Vector Machine (SVM), Logistic Regression, Stochastic Gradient Descent, K-Nearest Neighbor, Multi-Layer Perceptron, and k-Means clustering for workload classification. Post classification, the significance of attributes in predicting the workload is identified. For GCT dataset, task duration attribute is remarked to be significant whereas for the BBT dataset CPU usage is inferred to be significant.

In GCT, based on the attribute values the workloads are characterized into background tasks, demon tasks and system tasks. System tasks have a high CPU utilization and memory, but low disk utilization; Demon tasks uses CPU, memory, and disk for a relatively long duration, but percentage of utilization is less. Background processes have a moderate usage of CPU, memory while the disk utilization figures are observed to be low.

Mishra et al. [3] proposed a method for characterizing resource demands based on the classification of tasks in the Google's cloud back end. Different attributes considered for workload are task duration, CPU, and memory usage. The first step in the task classification is identifying the dimensions for the workload. In the second step, k-Means algorithm is used to cluster the tasks into different classes. In the third step, thresholds for the qualitative dimensions (small, large, and medium) are identified. In the last step, based on the range of dimensions, the tasks are combined.

Patel et al. [4] performed workload characterization using two clustering algorithms namely k-Means and Gaussian Mixture Model (GMM). Experiments are conducted on BBT and GCT workload traces. From the experiments, it is observed that GMM model better represents the heterogeneity in resource usage patterns with distinct cluster boundaries. It is assumed that the workload follows normal distribution, but there is no guarantee that workload always follows normal distribution. k-Means does not handle uncertainty when a data point is equi-distant to more than one cluster centroid. If a data point is arbitrarily assigned to a cluster, it was observed that the model was unable to represent the resource usage patterns properly.

From the literature, it is observed that limited research works exist for cloud workload characterization. Among the existing methods, most of the methods model the workloads by analyzing the statistical properties of the workload trace; however, studying the statistics alone may not be sufficient to model the dynamic behavior of workloads. Hence, in order to reproduce the dynamic behavior of the workload trace, we have chosen a double embedded stochastic process for modeling cloud workloads. Table 1 provides a summary of the existing research attempts in

Table 1 Existing works on workload characterization

Paper	Approach(s)	Workload Trace	Demerits
Robert et al. [14]	Statistical methods	GCT	(i) Workload characterization is performed at the machine level (ii) Characterization is done using basic statistics like mean and standard deviation
Patel et al. [4]	k-Means and GMM	BBT, GCT	(i) k-Means fails to handle heterogeneity (ii) GMM assumes that the workload trace follows normal distribution which is not always true
Shen et al. [1]	Statistical methods	BBT	(i) Resource usages are analyzed using statistics (ii) The proposed method is a trace-based method and its dependent on the system on which trace is recorded
Mishra et al. [3]	Statistical methods	GCT	Workload characterization is performed in terms of actual computational work but the underlying task domain distribution is not analyzed
Jung et al. [15]	Statistical methods	RUBiS	The proposed model is not suitable for the applications that are sensitive to performance

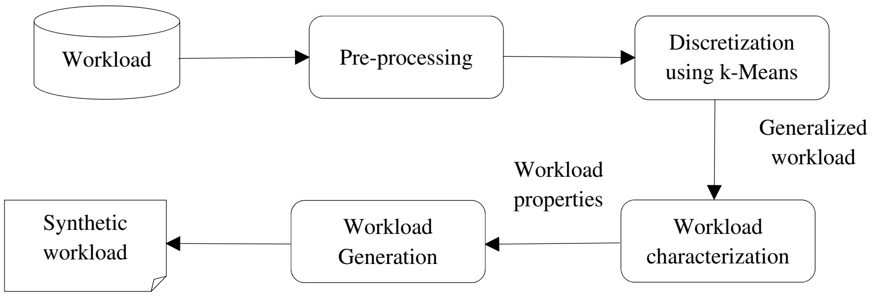

Fig. 1 Architecture of the Proposed Model

cloud workload characterization, approaches used by the existing works, and their demerits.

3 Proposed method

In this section, we describe the methodology adopted for workload characterization. The overall architecture of the proposed method is shown in Fig. 1.

3.1 Problem formulation

Let $M = \{VM_1, VM_2, .., VM_m\}$ be the set of Virtual Machines present in the data center, $R = \{R_1, R_2, .., R_n\}$ be the set of input requests arriving at the data center, and let each request R_i be described by a set of workload features $F_i = \{f_1, f_2, .., f_w\}$, where w may be different for different requests and $|F_i|$ denotes the number of resources needed by a specific request. Our objective is to design and evaluate a model which characterizes the workload based on the resource demands imposed by the requests to the applications running on the VMs. The model should characterize the workload generation as a double-embedded stochastic process [16] to capture the nature of the applications and tasks that have generated the workload. A double embedded stochastic process is a generative model that models R in two stages. In first stage, each request is modeled as a distribution over different task domains and in the second stage each task domain is modeled as a distribution over workload features F_i. A follow-up research objective is to assess the descriptive and predictive accuracy of the model using tasks such as synthetic workload generation, and comparing the model-generated task labels with actual task labels from which the request is generated.

3.2 Phases in the proposed methodology

The proposed workload characterization method broadly involves learning the model and evaluating the model's accuracy. The model learning and assessment steps comprise the following phases:

(i) Pre-processing: Pre-processing renders the input data to be in the required format for consumption by the proposed model.
(ii) K-bin discretization: Discretization groups similar data items into bins and subsequently aids in learning a generative model which is representative of the real workloads.
(iii) Workload characterization: In this phase, we develop an unsupervised model that characterizes the workload and models the application behavior that generated the workload.
(iv) Synthetic workload generation: After characterizing the workload, a workload can be generated by using the proposed model to assess the model's representativeness.

3.3 Pre-processing

Pre-processing transforms the workload into a format that a model can process efficiently. In this paper, our aim is to develop a workload characterization model which models the resource request and usage patterns.

In this paper, we have used two workload traces namely Bit Brains Trace and Alibaba Cluster Trace. For Bit Brains Trace, pre-processing involves removing invalid and duplicate requests and assigning an identifier for each request. A request is invalid if most of its features consist of NULL values. All the invalid requests are identified and removed. In the next step, each request is assigned a unique identifier for further processing.

For Alibaba Trace, the resource information is maintained across different tables. Therefore, in this case, we pre-processed the data by removing NULL values and extracted the required workload features by executing a join operation on the required tables.

The pre-processing of Alibaba Trace involves the following steps:

(i) We have removed the NULL values across the tables.
(ii) The records that are associated with the application name are extracted.
(iii) To obtain the resource usage at the instance level, a join operation is performed between the data extracted in step (ii) and the instance table in the workload trace.
(iv) From step (iii), the instance IDs are extracted, to get the resource metrics corresponding to each instance, the results obtained in step (iii) are joined with the records in the sensor table.

3.4 k-bin discretization

Instead of learning a workload characterization model with numerical workload parameters, we grouped the numerical values into discrete ranges that helps in avoiding overfitting and ease of inference of higher-order generative patterns. This discretization renders the model regularized and generalizable to unseen but related patterns in the workload. The proposed method uses k-Means based discretization method for discretizing the workload trace, which assumes a convex-bias and groups the data based on distance between the data points in the workload trace. In this work, we have used the Euclidean distance measure to calculate the distance between data points in a workload trace. Here discretization is applied on each feature i.e. k-Means is applied on each feature separately. Each workload feature is divided into *k* bins where the value of *k* depends on the data values present in each feature. Algorithm 1 outlines the discretization process.

```
Algorithm1: Discretization
Input: Workload Trace ($R$)
Output: Discretized Workload Trace
 1.  for each $f_i \in F$ do
 2.      count(i) ← unique($f_i$)
 3.      num_of_bins($f_i$) ← $\lceil \log_2 count(i) \rceil + 1$
 4.      $C_i$ ← kMeans($f_i$, num_of_bins($f_i$))
 5.      for each cluster $C_i$
 6.          for each data point $d \in C_i$
 7.              Replace the data point with the cluster number i
 8.          end for
 9.      end for
10.  end for
11.  return discretized workload trace
```

To decide the number of bins, we have used the Sturge's formula [17] as specified in Eq. (1), as it is suitable for dealing with different data distributions.

$$num\_of\_bins(f_i) = \lceil \log_2 unique(f_i) \rceil + 1 \tag{1}$$

where *num_of_bins*(f_i) represents the number of bins for the i^{th} workload feature and *unique*(f_i) is the number of unique values in the i^{th} feature. After deciding the number of bins for each feature, the *k*-Means discretization technique is applied. Subsequently, the data points in each workload feature are replaced with their corresponding bin numbers.

3.5 Workload characterization using probabilistic generative model

After discretization, each request in the trace consists of discrete values and some requests may be redundant because we have grouped the data into bins. The redundant requests are removed before characterizing the workload.

The proposed model is a generative probabilistic model which takes a set of requests created after applying discretization as an input and identifies the underlying task domain that generates a request i.e. whether the task is computationally intensive, web service-related task, or database related task. For example, a request to any Software as a Service (SaaS) suite may involve either compute-intensive or disk-intensive tasks, sometimes requests may be a mix of both compute-intensive and disk-intensive tasks. Disk-intensive tasks are likely to be spawned due to

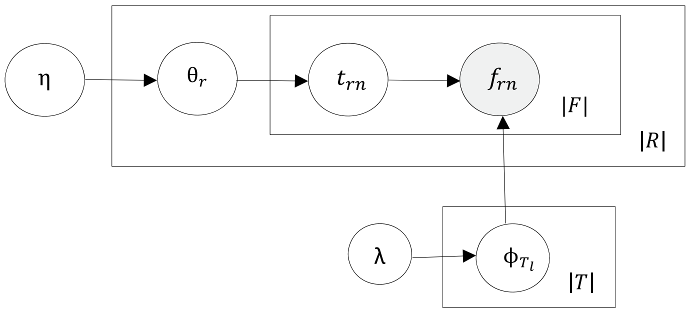

Fig. 2 Workload characterization using a probabilistic generative model

requests initiated from applications like storage, mail, and video streaming portals. Similarly, compute-intensive tasks are likely to be spawned by applications running simulations, virtual labs, or analytical models.

In the proposed model, instead of representing the request in terms of workload features (high dimension), each request is described as mixture of task domains (low dimension). Resource request and resource utilization may vary according to the underlying task domain, so, the proposed characterization model handles the heterogeneity. The generative model used for workload characterization is shown in Fig. 2.

The generative model shown in Fig. 2 is a plate diagram, which is commonly used to model the repetition of probabilistic conditional dependencies and independences between the latent and the observed random variates of the interest. Figure 2 illustrates that each workload feature in every request is analyzed and each request is expressed in terms of its originating task domain.

In Fig. 2, each circle denotes a latent variable; a shaded circle represents an observed variable, and each edge denotes the dependency between the variables connected by the edge. Plate denotes repetition of structures; for instance, $|R|$ in the plate denotes the cardinality of the repetition of the outer plate in Fig. 2.

The request-task domain distribution (θ_r) is a multinomial distribution representing mixture proportion of task domains from which a request 'r' is drawn. R denotes the collection of requests arriving at the data center and $\left|F_i\right|$ represents the number of features that describes a request. Task domain-feature distribution (ϕ_{T_l}) represents the feature distribution for a task domain T_l which is drawn from the Dirichlet prior λ, t_{rn} represents the task domain associated with n^{th} workload feature in a request 'r' and f_{rn} represents the n^{th} workload feature in a request 'r'.

Let $T = \left\{T_1, T_2, .., T_q\right\}$ be the set of task domains from which requests arrive at a data center. From the workload trace, only the workload feature values which describe the request are observed but task domain is not observed. Workload (request) is modeled as a distribution over a mixture of the latent task domains and each latent task domain is modeled as a distribution over the observable workload features. The assumptions made for this model are: Each request is generated from few task domains but not from all the task domains, similarly, each task domain consists of few workload features. In order to model the above-stated assumptions, we have chosen Dirichlet prior because of the following reasons:

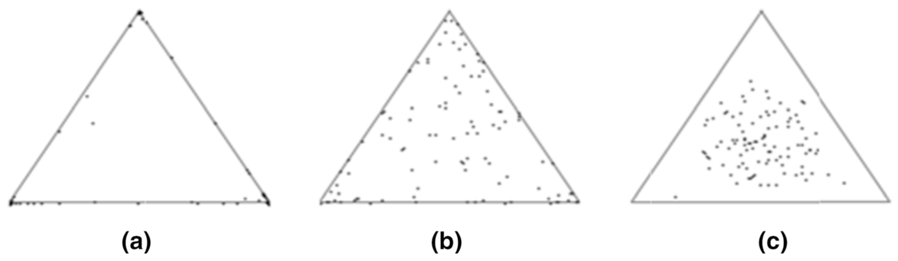

Fig. 3 Samples from the Dirichlet distribution when **a** $\eta = 0.05$ **b** $\eta = 0.5$ **c** $\eta = 3$

(i) Dirichlet is a conjugate prior for discrete distributions so that it simplifies the mathematical calculations. (ii) Dirichlet distribution can encode the sparsity i.e., it can encode the intuition that the requests are generated from few task domains.

In Fig. 2, θ_r represents each request as a proportion of task domains and θ_r is sampled from a Dirichlet distribution with a hyperparameter η and the probability density function is defined as in Eq. (2). $\Gamma()$ denotes the gamma function [18].

$$p(\theta_r|\eta) = \frac{\Gamma\left(\sum_{i=1}^{|T|}\eta_i\right)}{\prod_{i=1}^{|T|}\Gamma(\eta_i)}\prod_{i=1}^{|T|}\theta_{ri}^{\eta_i-1} \tag{2}$$

Dirichlet distribution [5] is parameterized by vector η whose length is the same as θ_r. Now, the samples drawn are controlled by η. Depending on the workload trace, the value of η may vary.

For instance, assume that the requests arriving at a data center are generated from three different task domains. Dirichlet distribution over three dimensions forms a simplex in 2-Dimensional space. Figure 3, shows how the Dirichlet parameter encodes sparsity in the underlying distribution. Each corner represents the task domain and a sample drawn from the simplex represents the proportion of task domains in a request.

From Fig. 3 it is observed that, for the values of η less than 1, the distribution has more mass on the corners and edges of simplex than in the center of the simplex. Hence, drawing the samples from a distribution (shown in 3(b)) allows us to model the request as a mixture of few task domains. Similarly, a task domain is modeled as a distribution over few workload features. From Fig. 2, it is observed that ϕ_{T_l} denotes the workload feature proportion in the task domain T_l and is given in Eq. (3).

$$p(\phi_{T_l}|\lambda) = \frac{\Gamma\left(\sum_{i=1}^{|F|}\lambda_i\right)}{\prod_{i=1}^{|F|}\Gamma(\lambda_i)}\prod_{i=1}^{|F|}\phi_{T_l i}^{\lambda_i-1} \tag{3}$$

From Fig. 2 it can be remarked that only workload features are observed and Bayesian inference [19] can be used to estimate the posterior distribution over the latent variables given workload features as given in Eq. (4). The objective is to learn the distributions namely, request-task domain distribution and task domain-workload feature distribution.

$$p(T,\theta,\phi|F) = \frac{p(F|T,\theta,\phi)p(T,\theta,\phi)}{\sum\sum\sum_T p(T,\theta,\phi,F)} \tag{4}$$

From the product rule, $p(F|T,\theta,\phi)\cdot p(T,\theta,\phi)$ is equivalent to the joint distribution $p(T,\theta,\phi,F)$. The joint distribution can be factorized as shown in Eq. (5), by observing the conditional dependencies from the generative model in Fig. 2.

$$p(T,\theta,\phi,F) = p(\theta|\eta)\cdot p(T|\theta)\cdot p(\phi|\lambda)\cdot p(F|T,\phi) \tag{5}$$

$$p(T,\theta,\phi,F) = \prod_{r=1}^{|R|} p\left(\theta_r|\eta_r\right)\prod_{r=1}^{|R|}\prod_{i=1}^{|F|} p\left(t_{ri}|\theta_r\right)\prod_{k=1}^{|T|} p\left(\phi_k|\lambda_k\right)\prod_{r=1}^{|R|}\prod_{i=1}^{|F|} p\left(f_{ri}\middle|t_{ri,\phi}\right) \tag{6}$$

Equation (6), illustrates the generative model presented in Fig. 2. Initially, each workload feature in each request is assigned with a task domain $\left(t_{rn}\right)$, task domain is a sample drawn from request-task domain distribution $\left(\theta_r\right)$ which is a multinomial distribution (discrete distribution). Once we know the task domain, we can infer the workload features that describe the task domain, from the task domain–workload feature distribution. This can be analogized with topic models [5] where a document is modeled as a distribution over topics and each topic is a distribution over words in a vocabulary.

Considering the first two terms from the right-hand side of Eq. (5).

$$\begin{aligned} p(\theta|\eta)p(T|\theta) &= \prod_{r=1}^{|R|} Dir\left(\theta_r|\eta_r\right)\prod_{r=1}^{|R|}\prod_{i=1}^{|F|}\prod_{k=1}^{|T|}\theta_{rt}^{t_{rik}} \\ &= \left(\left(\prod_{r=1}^{|R|}\left(\frac{\Gamma\left(\sum_{k=1}^{|T|}\eta_i\right)}{\prod_{k=1}^{|T|}\Gamma\left(\eta_i\right)}\right)\prod_{k=1}^{|T|}\theta_{rk}^{\eta_{rk}-1}\right)\right)\left(\prod_{r=1}^{|R|}\prod_{i=1}^{|F|}\prod_{k=1}^{|T|}\theta_{rk}^{t_{rik}}\right) \\ &= \left(\prod_{r=1}^{|R|}\left(\frac{\Gamma\left(\sum_{k=1}^{|T|}\eta_i\right)}{\prod_{k=1}^{|T|}\Gamma\left(\eta_i\right)}\prod_{k=1}^{|T|}\theta_{rk}^{\eta_{rk}-1+n_{rk.}}\right)\right) \end{aligned} \tag{7}$$

Equation (7) represents the number of workload features that are assigned to each task domain in the workload trace and $n_{rk.}$ represents how often a workload feature is assigned with a task domain t in the request r. Now, considering the last two terms on the right-hand side of Eq. (5), we obtain,

$$
\begin{aligned}
p(\phi|\lambda)p(F|T,\phi) &= \prod_{k=1}^{|T|} Dir\left(\phi_k|\lambda_k\right) \prod_{r=1}^{|R|}\prod_{i=1}^{|F|} p\left(f_{ri}|c_{ri,\phi}\right) \\
&= \left(\prod_{k=1}^{|T|}\left(\frac{\Gamma\left(\sum_{i=1}^{|F|}\lambda_{ki}\right)}{\prod_{i=1}^{|F|}\Gamma\left(\lambda_{ki}\right)}\prod_{i=1}^{|F|}\phi_{ki}^{\lambda_{ki}-1}\right)\right)\left(\prod_{r=1}^{|R|}\prod_{i=1}^{|F|}\left(\prod_{k=1}^{|T|}\phi_{kr_{ri}}^{c_{rik}}\right)\right) \\
&= \left(\prod_{k=1}^{|T|}\left(\frac{\Gamma\left(\sum_{i=1}^{|F|}\lambda_{ki}\right)}{\prod_{i=1}^{|F|}\Gamma\left(\lambda_{ki}\right)}\prod_{i=1}^{|F|}\phi_{ki}^{\lambda_{ki}-1+n_{.kv}}\right)\right)
\end{aligned}
\tag{8}
$$

where $|F|$ represents the set of workload features from all the requests in the workload trace and $n_{.kv}$ represents the number of times a workload feature is assigned to a task domain in any request. From Eqs. (7) and (8), the joint distribution in Eq. (5) can be written as

$$
p(T,\theta,\phi,F) = \left(\prod_{r=1}^{|R|}\left(\frac{\Gamma\left(\sum_{k=1}^{|T|}\eta_i\right)}{\prod_{k=1}^{|T|}\Gamma\left(\eta_i\right)}\prod_{k=1}^{|T|}\theta_{rk}^{\eta_{rk}-1+n_{rk.}}\right)\right)\left(\prod_{k=1}^{|T|}\left(\frac{\Gamma\left(\sum_{i=1}^{|F|}\lambda_{ti}\right)}{\prod_{i=1}^{|F|}\Gamma\left(\lambda_{ti}\right)}\prod_{k=1}^{|F|}\phi_{ki}^{\lambda_{ki}-1+n_{.kv}}\right)\right)
\tag{9}
$$

In order to compute the posterior distribution as specified in Eq. (4), we have now calculated the joint distribution which is in the numerator in Eq. (4) and now we need to calculate the marginal likelihood. To calculate marginal likelihood, we are integrating out the latent variables, but from Eq. (9) it is observed that it is difficult to integrate out t, because it is hidden in the variable n. This marginalization is done according to the process mentioned in [5, 22]. From Fig. 2 it is observed that, when we are conditioning on observed data, then latent variables are conditionally dependent on each other.

Due to the intractability in calculating the marginal likelihood, estimation of posterior distribution cannot be done analytically. To overcome this difficulty, the proposed model uses Gibbs Sampling which is a specific form of Markov Chain Monte Carlo (MCMC) process. MCMC is a sampling algorithm in which each random sample generated is used to generate the next random sample i.e. generating a sample from the posterior distribution depends on the current random sample [20, 21]. The proposed model uses Gibbs sampling because it is easy to implement with a low memory requirement, and converges to the posterior distribution. Gibbs sampling preserves the conditional dependencies [22] among the latent variables while approximating the posterior.

3.5.1 Approximating inference

This section explains how the Gibbs sampling approximates the posterior distribution. For instance, if we have a task domain-workload feature assignment t, then we can estimate proportion of task domain in a request (θ) and proportion of workload feature in a task domain (ϕ). Gibbs sampling considers each workload feature from the workload trace and estimates the probability of assigning workload feature to each task domain, conditioned on the other workload features.

Initially, each workload feature in each request is randomly assigned to a task domain. Gibbs sampler estimates the probability of associating the task domain $T_i \in T$ to the i^{th} feature in a request, by assuming that the associations of the remaining features with task domains are fixed. That is, while identifying the task-feature association, correlation between features in a request are considered. The estimation of the likelihoods of task-feature associations is done using Eq. (10) [23]. Algorithm 2 outlines Gibbs sampling and parameter estimation.

Algorithm 2: Parameter Estimation

```
Input: workload features $f$ where $f \in r$, count matrices $N^1_{|R|\times|T|}$ and $N^2_{|V|\times|T|}$
Output: Task domain assignment to workload features, θ, and ϕ
1.  Randomly initialize each workload feature with a task domain
2.  for each iteration do
3.      for $i = 0 \rightarrow | F | -1$
4.          $workload\_feature = f_i$
5.          $task\_domain = t_i$
6.          for $t = 0 \rightarrow |T| - 1$
7.              calculate the probability of assigning a workload feature $i$ to a
                    task domain $t$ using $p(t_i = t \mid t_{-i}, F, \eta, \lambda)$
8.          end for
9.          $task\_domain = samplefromp(t \mid t_{-i}, F, \eta, \lambda)$
10.     end for
11. assign the task domain selected in the step 9 to the workload feature $f_i$
12. update the count matrices $N^{(1)}$ and $N^{(2)}$
13. end for
14. calculate request-task domain distribution (θ) and task domain-workload feature distribution (ϕ)
15. $\theta_{rj} = \frac{N^{(1)}_{r,j}+\eta}{\sum_{k=1}^{|T|} N^{(1)}_{r,k}+T\eta}$
16. $\phi_{jf} = \frac{N^{(2)}_{f,j}+\lambda}{\sum_{f=1}^{|F|} N^{(2)}_{f,j}+|F|\lambda}$
```

Assigning a task domain to a workload feature depends on the following:

(i) The likelihood of the task domain j to be assigned to a request $r \in R$, which can be calculated as the fraction of features in r that are assigned to the task domain j.

(ii) The likelihood of observing feature i in the task domain j, which can be estimated using the proportionality given in Eq. (10).

$$p(t_i = j|t_{-i}, F, \eta, \lambda) \propto \frac{N^{(2)}_{f,j} + \lambda}{\sum_{f=1}^{|F|} N^{(2)}_{f,j} + |F|\lambda} \cdot \frac{N^{(1)}_{r,j} + \eta}{\sum_{t=1}^{|T|} N^{(1)}_{r,t} + T\eta} \tag{10}$$

where $N^{(1)} and N^{(2)}$ are the count matrices of dimensions $|R| \times |T|$ and $|F| \times |T|$ respectively. $N^{(2)}_{f,j}$ is the number of times a workload feature f is assigned to the task domain j and $N^{(1)}_{r,j}$ is the number of times a task domain j is assigned to workload features in a request r. Algorithm 2 outlines Gibbs sampling and parameter estimation.

Equation (10) computes a probability vector that represents the likelihood of each workload feature to belong to each of the task domains. Based on the task domain associated with each feature in the request, a workload can be characterized.

Each iteration mentioned in the step 2 of the Algorithm 2 generates a sample that consists of task domain assignment to each workload feature in the workload trace. Initial samples generated from the Gibbs sampler will not estimate the posterior distribution. The samples which approximate the posterior distribution are saved and used to approximate θ and ϕ.

3.6 Synthetic workload generation

Once the characterization model is built, it can be used to generate synthetic workloads. After building the characterization model, the model parameters namely number of task domains (|T|), η and λ are known. To generate a synthetic workload, workload feature distribution is drawn for each task domain. Let M be the number of requests to be generated and N be the length of each request. For each request, a distribution over task domains is inferred. Subsequently, workload features in each request can be generated by first sampling a task domain for the feature from the task-domain distribution and then sampling a specific value for the feature from the workload-feature distribution. The process of Synthetic Workload Generation is outlined in Algorithm 3. We map the workload feature label that is generated to its original numerical value, using the following method:

(i) The name of the workload feature can be used to identify the bin from which this feature is potentially generated.
(ii) Based on the boundaries of the bin, the values from the workload trace which fall into the bin are considered.
(iii) The distribution that the data follows and the parameters of the distribution are then inferred.
(iv) A workload value is sampled from the distribution.

Algorithm 3: Synthetic Workload Generation

```
Input: workload characterization model $£ = (\eta, \lambda, |T|)$
Output: workload trace
1. for $l = 1$ to $|T|$
2.     sample the parameters of the workload feature distribution for each task
       domain $T_l$ as $\phi_{T_l} \sim Dirichlet(\lambda)$
3. end for
4. for $r = 1$ to $M$
5.     sample parameters for request-task domain distribution $\theta_r \sim Dirichlet(\eta)$
6.     for each workload feature $f_i$=1 to N
7.          select the task domain for the workload feature $T_l \sim Multinomial(\theta_r)$
8.      end for
9.     Generate a workload feature in a trace from a task domain - workload feature
       distribution $f_i \sim Multinomial(\phi_{T_l})$
10. end for
```

4 Experimental setup and evaluation of the proposed algorithm

In this section, we describe the experiments carried out for evaluating the proposed workload characterization model. Experiments are conducted on a 12-core Intel (R) Xeon (R) W-2265 series processor with 12 cores running at 3.50 GHz with 32 GB RAM. We have used free open-source packages like pandas,[1] ldamodel,[2] stat models,[3] and matplotlib[4] available in python for implementing the proposed method.

4.1 Dataset

To perform our experiments, we have used the Alibaba Cluster Workload Trace [24] and GWA-T-12 Bit Brains [1] workload trace.

In order to fulfill the computing demands of Machine Learning (ML) workloads, Alibaba cloud offers Machine Learning Platform for AI (PAI). Alibaba cluster has 1295 2-GPU machines and 519 8-GPU machines. CPU-to-GPU ratio is less in machines with 8 GPUs when compared to the machines with 2 GPUs. Alibaba cluster has released three versions of cluster traces. In our work, we have used the latest version of the workload trace released in the year 2020. PAI provides various services covering tasks like feature engineering, training, evaluation, and inference. Figure 4 shows the architecture of the PAI.

1 https://pandas.pydata.org.
2 https://radimrehurek.com/gensim/models/ldamodel.html.
3 https://www.statsmodels.org/stable/index.html.
4 https://matplotlib.org.

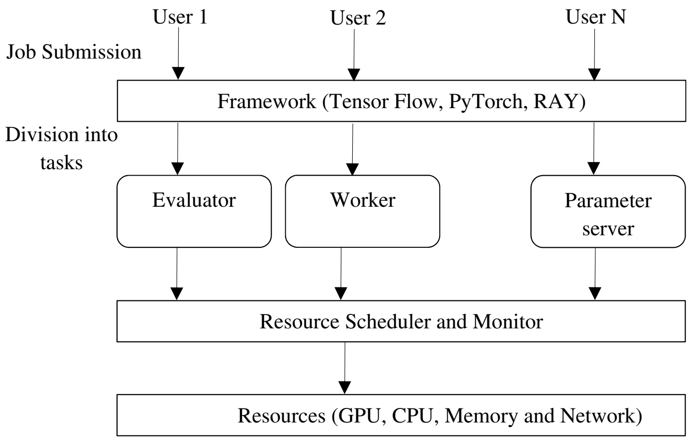
User 1
User 2
User N
Job Submission
Framework (Tensor Flow, PyTorch, RAY)
Division into tasks
Evaluator
Worker
Parameter server
Resource Scheduler and Monitor
Resources (GPU, CPU, Memory and Network)

Fig. 4 Framework of PAI

Alibaba Trace comprises the information about the training and inference jobs running Machine Learning algorithms in various frameworks [25–27]. When the user submits a job, the user provides an application code along with a request for the amount of resources required for executing the job. The trace records the resource requests and resource usage of workload at multiple levels namely at job level, task level, and instance level. Trace also records the submission and completion time. From the trace, it is observed that some workloads in the trace are labeled with application names.

The different applications found in the trace are Bidirectional Encoder Representations from Transformers (BERT), Neural Machine Translation (NMT), Clique Through Rate prediction (CTR), graph learn, inception, and Residual Neural Network (ResNet). Trace also includes the specifications of the machines present in the cluster. For our experiments, we have used the workloads which are associated with application names. In the trace, we have 890,755 records that are associated with application names.

The trace consists of the tables namely pai_job_table, pai_task_table, pai_instance_table, pai_sensor_table, pai_group_tag_table, pai_machine_spec and pai_machine_metric_table. From the tables, with the help of a join operation, we have extracted the necessary workload features that describes resource requests and resource usage patterns of jobs submitted to PAI.[5]

GWA-T-12 Bit Brains Trace contains the performance metrics of 1750 Virtual Machines from a distributed data center. The different features in the workload trace are CPU cores, CPU capacity provisioned [MHZ], CPU usage [MHZ], CPU usage [%], Memory capacity provisioned, Memory usage [KB], Disk read throughput [KB/s], Disk write throughput [KB/s], Network received throughput [KB/s], and Network transmitted throughput [KB/s]. Table 2 shows the statistics of the workload

[5] https://github.com/sindhu1018/alibaba_workload_trace.

Table 2 Workload Trace Statistics

S.No	Workload Trace	Period of data collection	Number of VMs
1	BBT	4 months	1750
2	Alibaba	2 months	1814 (6742 GPUs)

Table 3 Alibaba Cluster Workload Trace

	Mean	Median	Standard Deviation	Width of the Range
Plan_cpu	514.29	600	356.91	6399
Plan_mem	18.52	19.53	15.64	292.87
Plan_gpu	59.63	50.00	54.95	799
GPU_wrk_util	11.71	4.84	20.57	423.20
CPU_usage	201.89	96.33	373.19	8673.34
Avg_mem	4.02	2.56	5.90	240.34
Max_mem	7.48	3.49	15.81	2262.83
Avg_gpu_wrk_mem	1.37	0.54	2.66	147.75
Max_gpu_wrk_mem	2.25	0.76	3.31	151.10

traces used in our experiments and the detailed statistics of Alibaba PAI Trace and BBT workload trace are shown in Tables 3 and 4.

In Table 3, plan_cpu represents the number of CPU cores requested in percentage, plan_mem is the amount of main memory requested in GB, plan_gpu denotes the number of GPUs requested in percentage, gpu_wrk_util denotes the number of GPUs used in percentage, cpu_usage denotes the number of CPU cores used in percentage, avg_mem represents in GB the main memory used in average, max_mem represents the maximum main memory used in terms of GB, avg_gpu_wrk_mem represents in GB the GPU memory used in average and max_gpu_wrk_mem denotes the maximum GPU memory used in GB.

4.2 Discretization

In this section, we describe the discretization procedure adopted for BBT workload trace and the same procedure is followed for Alibaba cluster workload trace. In BBT each request for application is described by ten features. After preprocessing, discretization is performed as described in Algorithm 1. The number of bins and ranges for each feature for the BBT trace are as shown in Table 5. For the Alibaba trace, the number of bins created for the features plan_cpu, plan_gpu, plan_mem, cpu_usage, gpu_wrk_util, avg_mem, and max_mem is 3,4,6,15,9,10, and 9 respectively. From Table 5, it is observed that the number of bins for the feature CPU core is 3 and ranges of bins are:

Bin 1: $0 \leq X < 9$.

Bin 2: $9 \leq X < 24$.

Table 4 BBT workload trace

	CPU capacity provisioned [MHZ]	CPU usage [MHZ]	Memory capacity provisioned [KB]	Memory usage [KB]	Disk read throughput [KB/s]	Disk write throughput [KB/s]
Mean	8116.08	1260.54	10,450,438.29	9223.37	335.58	74.19
Median	5199.99	58.52	92,233.72	9223.37	0	1.66
Standard Deviation	9223.37	4280.69	29,921,629.63	9223.37	5180.48	1109.72
Range	86,399.98	63,876.55	536,309,760	402,653,184	922.33	922.33

Table 5 Bins created after discretization

S.No	Workload Feature	Number of bins	Range
1	CPU cores	3	[0, 9, 24, 32]
2	CPU capacity provisioned [MHZ]	9	[0, 1371.63, 4039.96, 8042.81, 13, 174.48, 18, 307.12, 31, 484.25, 64, 177.10, 86, 399.95, 86, 399.98]
3	CPU usage [MHZ]	11	[0, 1924.85, 6977.96, 13, 243.59, 18, 684.74, 22, 525.28, 29, 243.75, 38, 656.67, 48, 185.05, 54, 558.05, 57, 075.73, 63, 876.55]
4	CPU usage [%]	12	[0, 4.97, 15.39, 29.45, 45.09, 61.11, 76.64, 89.72, 99.09, 106.37, 116.62, 155.01, 186.6]
5	Memory provisioned (KB)	15	[0, 7.31437669e+06, 1.70144500e+07, 2.82717793e+07, 4.34680750e+07, 5.80776385e+07, 6.56506753e+07, 7.09023716e+07, 7.98663069e+07, 9.37039703e+07, 1.17239136e+08, 1.42424085e+08, 1.98993875e+08, 2.56999163e+08, 4.02638609e+08, 5.36309e+08]
6	Memory usage (KB)	13	[0, 7.44330544e+05, 2.78725776e+06, 5.66202518e+06, 9.02654164e+06, 1.30827728e+07, 2.16144024e+07, 4.15656509e+07, 7.09359450e+07, 1.44034859e+08, 2.47408011e+08, 3.29487314e+08, 3.84069190e+08, 4.02653184e+08]
7	Disk read throughput (KB/s)	2	[0, 45, 016.66, 1, 444, 406.93]
8	Disk write throughput (KB/s)	11	[0, 1464.81, 5411.26, 11, 870.96, 22, 772.54, 35, 450.73, 49, 867.68, 72, 617.96, 98, 565.79, 125, 746.19, 154, 903.44, 192, 405.80]
9	Network received throughput (KB/s)	8	[0, 5.8915e+02, 2.49349e+03, 6.26289e+03, 1.29106233e+04, 2.21903841e+04, 3.51783992e+04, 4.61106461e+05, 8.79122500e+05]
10	Network transmitted throughput (KB/s)	8	[0, 2.34624197e+03, 1.01901589e+04, 2.53912435e+04, 5.62198881e+04, 1.05572496e+05, 1.79424192e+05, 1.74733236e+06, 3.26959812e+06]

Bin 3: $24 \leq X \leq 31$.
Here X represents the CPU cores feature from BBT workload trace.

4.3 Workload characterization

4.3.1 Result analysis of BBT trace

After discretization, each value in the input request is replaced with a string "*bin number_workload feature*". Now, each request in the workload trace consists of a set of strings. The proposed model is trained with 80% of the requests and 20% are used for evaluating the performance of the model. In order to decide the number of task domains involved in generating the workload trace, coherence measure is used. Coherence measures the conditional likelihood of workload features in a task domain. Coherence score is calculated using Eqs. (11) and (12).

$$coherence\left(T_l\right) = \sum_{\left(f_i,f_j\right)\in T_l} score\left(f_i,f_j\right) \tag{11}$$

$$score\left(f_i,f_j\right) = log\frac{p\left(f_i,f_j\right)+ \in}{p\left(f_i\right) * \left(f_j\right)} \tag{12}$$

where T_l is the task domain, score is calculated for each task domain and then average value of score of all the task domains is taken as a coherence value. Coherence score of a task domain depends on the co-occurrence of workload features.

We have trained the proposed model by varying the values of the parameters namely η, λ, and $|T|$, and the corresponding coherence value is measured. The cardinality of T at which the coherence score is maximum is considered as number of task domains. If we have maximum coherence at different values, then to decide the best value for number of task domains we have calculated the perplexity. Perplexity measures the generative capability of the trained model and it is the inverse of geometric mean.

$$perplexity\left(R_{test}\right) = \exp\left\{-\frac{\sum_{i=1}^{M}\log p\left(F_i\right)}{\sum_{i=1}^{M}\left|F_i\right|}\right\} \tag{13}$$

where R_{test} represents the set of requests in the test data, F_i represents the feature values in ith request in test data, and $\left|F_i\right|$ represents the number of feature values in ith request. From Eq. (13), it is observed that perplexity is a decreasing function of the likelihood of unseen requests. The lower the perplexity, the better the model.

4.3.2 Effect of hyperparameters

To limit the number of possible combinations of η, λ, and$|T|$, we have set a permissible range of values for η, λ. This section outlines the selection of appropriate ranges for the hyperparameters η and λ. The parameter η represents request-task

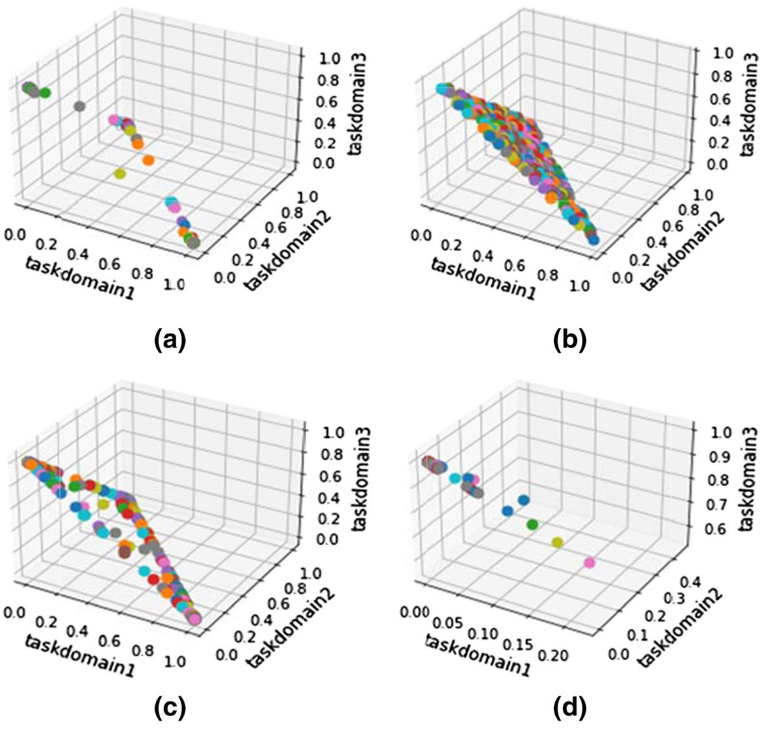

Fig. 5 Effect of the hyperparameter η **a** η = [0.01, 0.01, 0.01] **b** η = [1, 1, 1] (**c**) η = [0.1,0.1,0.1] **d** η = [0.02, 0.01, 5.0]

domain density and λ represents task domain-workload feature density. When η is small, it indicates that each request is generated from few task domains. If η is large, it indicates that more kinds of tasks are involved in generating a request. However, setting η to a large value may result in overlapped task domains; thereby facilitating the consolidation of different yet closely-related task domains.

In our experiments, we have chosen η from [0.01, 0.05, 0.1, 0.5] because typically, each request is generated from only one or two task domains. For instance, in Alibaba workload trace, each request is from a specific kind of Machine Learning task. So, we have chosen a small value for η. Figure 5 shows the effect of η, we have considered three task domains for interpretation. From Fig. 5 it is observed that when $\eta = 0.01$, the requests are concentrated to the boundaries indicating that each request is generated from a specific task domain. Similarly, when λ is small, task domain may consist of few workload feature values and when λ is large, each task domain contains most of the workload features. It is typically not possible to characterize a task domain with less workload features, and hence in our experiments we have chosen λ from [0.5, 1, 1.5, 2], thereby a request with a minimum workload feature also can be accurately characterized.

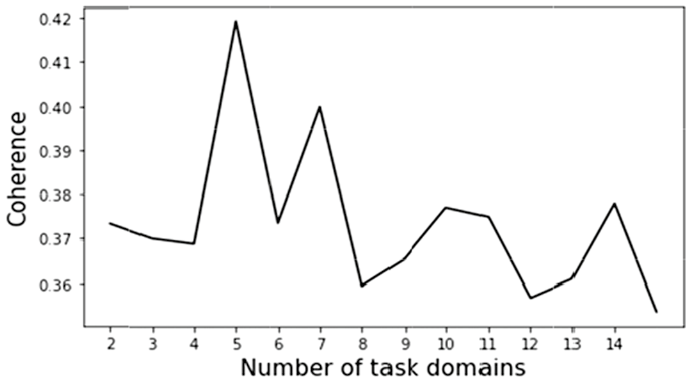

Fig. 6 Deciding the number of task domains for BBT trace

4.3.3 Identifying the number of task domains

To determine the number of task domains from which workload trace has been generated, coherence measure is used as given in Eq. (11) and Eq. (12). While trying to fix the number of task domains, we have varied the values of Dirichlet priors as explained in Sect. 4.3.1 to determine the influence of the Dirichlet priors on coherence. Finally, the value at which coherence is maximum is set as the number of task domains. Figure 5 shows the coherence values for different task domains.

From Fig. 6, for BBT it is observed that workload trace is generated from five categories of tasks and it has a peak value at T = 5, 7. so, we have validated the LDA model by calculating the perplexity and results are shown in Table 6. From Table 6, it is observed that the model with number of task domains as five, yields a lower perplexity, and the corresponding hyperparameter (η, λ) values are (0.5, 1).

After identifying the number of task domains, each workload feature is associated with a task domain, and task domains are labeled based on the distribution ϕ. Virtual Machines in the Bit Brains Trace host the applications related to business computations for enterprises. Therefore, the requests may be generated from tasks such as batch tasks, web service tasks, interactive tasks, and Online Transaction Processing tasks. Next, each request in the workload trace is assigned to a task domain based on the number of features assigned to a task domain. For example, if a request

Table 6 Perplexity of BBT trace

S.No	(η,λ)	*perplexity*($\|T\| = 5$)	Perplexity ($\|T\| = 7$)
1	(0.01, 0.01)	226.74	257.86
2	(0.05, 0.1)	185.91	195.32
3	(0.5, 1)	156.46	168.86
4	(0.01, 2)	253.01	312.24
5	(0.3, 1)	170.72	182.20
6	(0.01, 0.1)	209.67	223.54

Table 7 Task domains and their resource utilization levels

Task Domain	Utilization Levels
1	low_CPU usage, low_memory utilization, low_disk utilization, low_network utilization
2	medium_CPU usage, low_memory utilization, low_network utilization
3	high_CPU usage, high_memory utilization
4	low_CPU usage, low_memory utilization, high_disk utilization, high_network utilization
5	low_CPU usage, medium_memory utilization, low_network utilization

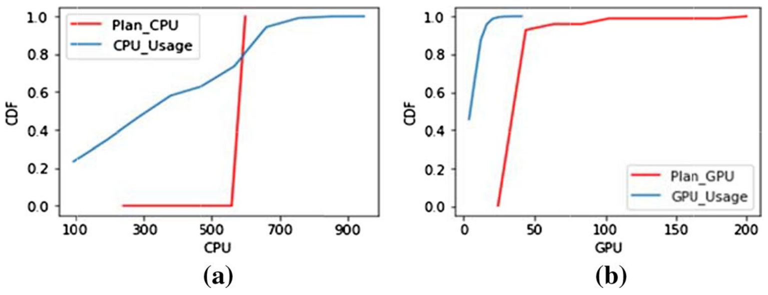

Fig. 7 Resource plan and usage of CTR instances **a** CPU **b** GPU

consists of most of the workload features from the task domain j, then the request is considered to have been generated from the task domain j. Table 7 shows the task domains and their resource utilization levels based on workload features associated with task domains.

Task domain 1, consists of the workload features {0_CPU usage [MHZ], 0_CPU capacity provisioned [MHZ], 1_CPU capacity provisioned [MHZ], 0_CPU usage [%], 0_Memory usage [KB], 0_Memory capacity provisioned [KB], 0_Disk read throughput (KB/s), 0_Network transmitted throughput [KB/s], 0_Network received throughput [KB/s], 0_Disk write throughput [KB/s]}.

From the workload features present in task domain 1, with respect to CPU utilization it is observed that task domain 1 consists of the requests where the maximum CPU utilization is 4.97% of the CPU resource provisioned (From Table 4). Similarly, task domain 1 consists of requests where the maximum CPU utilization is 106% of the CPU resource provisioned. This kind of characterization of requests with respect to resource utilization will help in avoiding resource over provisioning and under provisioning.

From Table 7, it is observed that if a request is generated from the task domain 1, then the request needs a low volume of the resources namely CPU, Memory, Disk, and Network. The requests that are generated from the task domain 3 are a mix of CPU and memory and their utilization levels are high.

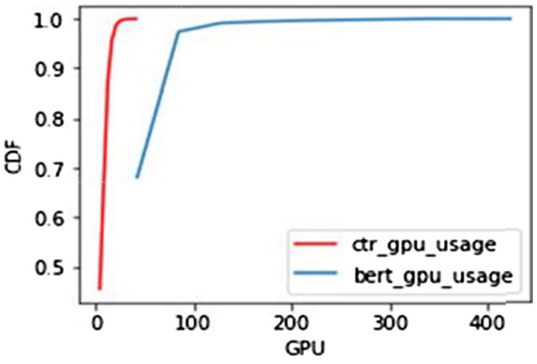

Fig. 8 GPU usage levels of CTR and BERT instances

4.4 Result analysis of Alibaba trace

In this section, the characteristics of tasks in the Alibaba trace are studied and compared with the characteristics generated by proposed model. Alibaba trace consists of resource request and usage patterns of different applications namely BERT, NMT, CTR, graph learn, inception and ResNet. Figure 7 shows the request and usage of resources namely CPU and GPU. From Fig. 7a, it is observed that 45% of CTR instances are using a greater number of CPU cores than requested and from Fig. 7b it is observed that GPU utilization is low. Figure 8 shows the comparison between GPU usage levels of CTR and BERT instances. From Fig. 8, it is observed that BERT instances use higher levels of GPU than CTR instances.

4.4.1 Identifying the task domains for Alibaba Trace

From the Fig. 9, it is observed that coherence value is maximum when the number of task domains is taken around 6 and 7. To resolve the tie, we have calculated the perplexity of the test data. The perplexity values at task domains 6 and 7 are 1004.07 and 1208.66 respectively. Therefore, in our experiments, for Alibaba workload trace the number of task domains are set to 6. After deciding the number of task domains, our model is applied to obtain the workload features in each task domain.

Table 8 shows the task domains and their resource request and utilization. From Table 8, it is observed that the task domain 1 consists of the workload features where the GPU utilization is low when compared with CPU utilization and from the original Alibaba Workload Trace, graphlearn application related instances have a low GPU utilization and High CPU utilization [28], therefore task domain1 can be labeled with graphlearn. Similarly, task domains 2, 3, 4, 5, and 6 are labeled with NMT, BERT, CTR, ResNet and inception respectively. For each feature, the utilization levels (low, medium or high) are decided based on the minimum and maximum values in the workload feature.

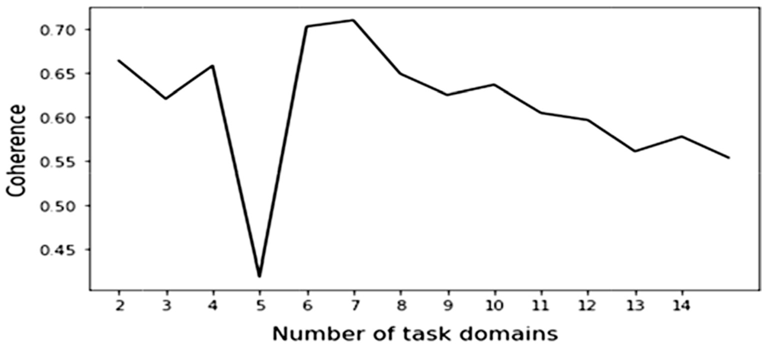

Fig. 9 Deciding the number of task domains for Alibaba Workload Trace

Table 8 Task domains and their resource utilization levels

Task Domain	Utilization Levels
1	low_GPU utilization, high_CPU utilization, medium_memory utilization
2	high_GPU utilization, high_CPU utilization, high_memory utilization
3	medium_GPU utilization, high_CPU utilization, medium_memory utilization
4	low_GPU utilization, medium_CPU utilization, high_memory utilization
5	high_GPU utilization, high_CPU utilization, medium_memory utilization
6	high_GPU utilization, low_CPU utilization, high_memory utilization

5 Evaluating synthetic workload

In this section, we present a comparison between the synthetic and real workload. The goal is to determine whether the synthetic workload reflects the real workload.

5.1 Comparing synthetic and real workload using correlation coefficient

Synthetic and real workloads are compared by calculating Pearson Correlation Coefficient (PCC) between the workload features. PCC results a value in the range [−1, 1]. If the correlation coefficient between the two features is zero then it indicates the features are unrelated. For BBT real workload, the correlation coefficient between CPU capacity provisioned and memory capacity provisioned is 0.68. Synthetic workload generated using the procedure mentioned in Sect. 3.6 is available in the GitHub repository.[6] The correlation coefficient between CPU capacity provisioned and memory capacity provisioned is 0.43. For real workload, the correlation coefficient between CPU capacity provisioned and CPU usage is 0.35. For synthetic

[6] https://github.com/sindhu1018.

Table 9 Distribution Fit of real workload trace

Workload Feature	Exponential	Lognormal	Gamma	Normal	Rayleigh
CPU capacity provisioned	**2111**	6149	4812	3241	2974
CPU usage	10,380	10,299	**10,125**	11,470	11,608
Memory capacity provisioned	3925	3526	**1836**	4850	4648
Memory Usage	18,235	**18,089**	18,242	19,683	19,359

Table 10 Distribution Fit of synthetic workload trace

Workload Feature	Exponential	Lognormal	Gamma	Normal	Rayleigh
CPU capacity provisioned	**11,650**	13,500	11,668	12,092	11,939
CPU usage	11,229	12,503	**11,046**	11,769	11,202
Memory capacity provisioned	20,750	23,254	**20,738**	21,215	21,048
Memory Usage	19,505	**19,438**	19,484	21,354	21,101

workload, the correlation coefficient between CPU capacity provisioned and CPU usage is 0.39.

From the PCC measure, it is observed that the workload features namely, *CPU capacity provisioned* and *memory capacity provisioned* are positively correlated in both real workload trace and synthetic workload trace. Similarly, *CPU capacity provisioned* and *CPU usage* are also positively correlated in both real workload trace and synthetic workload trace.

5.2 Comparing synthetic and real workload using distribution fit

To compare synthetic and real workload trace, we have fitted different distributions and we have selected the distributions that fits the data better, based on Akaike Information Criterion (AIC) [29]. AIC estimates the relative amount of information loss of a given model. Therefore, the distribution with a low AIC score will fit the data better. AIC score is calculated using Eq. (14). Results of distribution fit are shown in Tables 8 and 9.

$$AIC = \left(\frac{2d}{d - s - 1}\right)k - 2\ln(L) \tag{14}$$

where d is the number of observations, s is the number of parameters and L is the maximum likelihood for the estimated model.

Table 9 shows the distributions that generate the BBT workload trace. Different workload features in the BBT trace follow different distributions. From Table 9 it is observed that the workload feature *CPU capacity provisioned* follows an exponential distribution because exponential distribution has the lowest AIC score when compared with the other distributions.

Table 11 Test Statistics

S.No	Quantity Under Test	Test Statistic
1	CPU capacity provisioned	0.1209
2	CPU usage	0.0581
3	Memory provisioned	0.8829
4	Memory usage	0.0719
5	Disk read throughput	0.0299
6	Disk write throughput	0.2949
7	Network received throughput	0.1933
8	Network transmitted throughput	0.3163

From Table 9 and Table 10, it is observed that the workload feature CPU usage from real workload trace and synthetic workload fits to gamma distribution, similarly, memory usage fits to lognormal distribution. From this, it is observed that workload features in both synthetic and real workload are generated from same probability distributions. In Tables 9 and 10, the bold values represent the distributions that model the corresponding workload features well.

5.3 Comparing synthetic and real workload using a statistical test

To prove that the synthetic workload generated by the proposed model is similar to the real workload trace, we have performed Wilcoxon test [30]. To perform Wilcoxon test, the null hypothesis ($\mathrm{H_0}$) and alternate hypothesis $\left(\mathrm{H_A}\right)$ is set-up as follows:

$$H_0 : M_{f_i}^R = M_{f_i}^S$$

$$H_A : M_{f_i}^R \neq M_{f_i}^S$$

where $M_{f_i}^R$ denotes the median of i^{th} workload feature in the real workload trace and $M_{f_i}^S$ denotes the median of i^{th} workload feature in synthetic workload trace.

Table 11 shows the statistical test results. Each row in the table describes the test statistic obtained by performing Wilcoxon's test between the workload features in real workload trace and synthetic workload trace. As seen in Table 11, the Wilcoxon tests do not provide sufficient evidence to reject the null hypothesis at 99% confidence level and hence there is sufficient statistical evidence to assert that the synthetic workload generated by the model is representative of the real workload trace.

6 Conclusion and future work

In this paper, we have proposed a workload characterization technique for Cloud based on a probabilistic graphical model. The proposed model overcomes the difficulties in handling heterogeneous requests arriving at a data center because the model can learn from an attribute set of varying lengths. In the proposed model, Latent Dirichlet Allocation is used and the proposed characterization model can be used for generating synthetic workloads that are representative of the real workload. The representativeness of the synthetic workload is quantified in two ways (i) using correlation metric (ii) identifying a distribution that best fits the data and (iii) by performing statistical tests. From the experiments it is observed that the correlation between the workload features in the real-world workload trace and in synthetic workload trace are similar, and also the distribution that fits real-world trace and that of the synthetic workload are similar. The proposed model is capable of identifying the properties of task domains that have generated the workload trace. In future, we are planning to develop a temporal workload characterization model and investigate its applicability to the tasks like future workload prediction, and task scheduling.

Acknowledgements The authors would like to thank Mr. Gaurav and Mr. Sukhdev Singh who are undergraduate students in National Institute of Technology Andhra Pradesh, for their help with dataset collection and experimental setup.

Funding This work has been sponsored by LinkedIn under a research grant for the project entitled "A Scalable Resource Requirement Prediction and Provisioning Framework for Elastic Cloud".

Data availability statement Real workload traces available at Bit Brains [1] and Alibaba cluster traces [24] have been used. The synthetic workloads generated from the proposed model are available in the GitHub repository that is accessible through https://github.com/sindhu1018.

References

1. Shen S, van Beek V, and Iosup A (2015) Statistical characterization of business-critical workloads hosted in cloud datacenters In proc 15th IEEE/ACM Int'l Symp. Cluster, Cloud, and Grid Computing (CCGRID 15)
2. Cano I, Aiyar S and Krishnamurthy A (2016) Characterizing private clouds: a large-scale empirical analysis of enterprise clusters In proc 7th ACM Symposium on Cloud Computing (SoCC 16)
3. Mishra A, Hellerstein J, Cirne W, Das C (2010) Towards characterizing cloud backend workloads: insights from google compute clusters. ACM SIGMETRICS Perform Eval Rev 37(4):34–41. https://doi.org/10.1145/1773394.1773400
4. Patel E, Kushwaha DS (2020) Clustering cloud workloads: K-Means vs Gaussian mixture model. Procedia Computer Sci 171:158–167. https://doi.org/10.1016/j.procs.2020.04.017
5. Blei D, Ng AY, Jordan MI (2003) Latent Dirichlet allocation. J Mach Learn Res 3:993–1022
6. Mahambre S, Kulkarni P, Bellur U, Chafle G and Deshpande D (2012) Workload characterization for capacity planning and performance management in IaaS cloud In IEEE International Conference on Cloud Computing in Emerging Markets (CCEM 12) https://doi.org/10.1109/CCEM.2012.6354624

7. Nemati H, Azhari SV, Shakeri M, Dagenais M (2021) Host-based virtual machine workload characterization using hypervisor trace mining. ACM Trans Model Perform Eval Comput Syst 6(1):1–25. https://doi.org/10.1145/3460197
8. Shishira S, Kandasamy A and Chandrasekaran K (2017) Workload characterization: survey of current approaches and research challenges In proc 7th International Conference on Computer and Communication Technology (ICCCT17)
9. Williams A, Arlitt M, Williamson C, Barker K (2005) Web workload characterization: ten years later. In: Tang X, Xu J, Chanson ST (eds) Web content delivery, vol 2. Springer, New York
10. Menasct A, D A, Almeida V, Fonseca R, and Mendes MA (1999) A methodology for workload characterization of E-commerce sites In proc ACM Conference on Electronic Commerce (EC 99)
11. Khan A, Yan X, Tao S, Anerousis N (2012) Workload characterization and prediction in the cloud: a multiple time series approach. IEEE Netw Oper Manag Symp. https://doi.org/10.1109/NOMS.2012.6212065
12. Aragon H, Braganza S, Boza EF, Parrales J, Abad CL (2019). Workload characterization of a software-as-a-service web application implemented with a microservices architecture In proc World Wide Web Conference (WWW '19)
13. Shekhawat VS, Gautam A and Thakrar A (2018) Datacenter workload classification and characterization: an empirical approach In proc IEEE 13th International Conference on Industrial and Information Systems (ICIIS 18)
14. Birke R, Chen LY, Smirni E (2014) Multi-resource characterization and their (in) dependencies in production datacenters In 2014 IEEE Network Operations and Management Symposium (NOMS) (pp 1–6) IEEE
15. Jung G, Sharma N, Goetz F, Mukherjee T (2013) Cloud capability estimation and recommendation in black-box environments using benchmark-based approximation In 2013 IEEE Sixth International Conference on Cloud Computing (pp 293–300) IEEE
16. Tsyrulnikov M, Rakitko A (2019) Impact of non-stationarity on hybrid ensemble filters: a study with a doubly stochastic advection-diffusion-decay model. Q J R Meteorol Soc 145(722):2255–2271
17. Sturges HA (1926) The choice of a class interval. J Am Stat Assoc 21(153):65–66. https://doi.org/10.1080/01621459.1926.10502161
18. Sebah P, Gourdon X (2002) Introduction to the gamma function. Am J Sci Res. https://www.csie.ntu.edu.tw/~b89089/link/gammaFunction.pdf. Accessed 10 June 2022
19. Box GEP, Tiao GC (1973) Bayesian inference in statistical analysis. Wiley, USA
20. van Ravenzwaaij D, Cassey P, Brown SD (2018) A simple introduction to Markov Chain Monte-Carlo sampling. Psychon Bull Rev 25:143–154. https://doi.org/10.3758/s13423-016-1015-8
21. Gilks WR, Richardson S, Spiegelhalter DJ (eds) (1996) Markov chain Monte Carlo in practice. Chapman & Hall/CRC, Boca Raton
22. Teh YW, Newman D and Welling M (2006) A collapsed variational bayesian inference algorithm for latent dirichlet allocation In proc. 19th International Conference on Neural Information Processing Systems Advances in Neural Information Processing Systems
23. Darling WM (2011) A theoretical and practical implementation tutorial on topic modeling and gibbs sampling-school of computer science. In: Proc.49th annual meeting of the association for computational linguistics: human language technologies
24. Alibaba production cluster data. https://github.com/alibaba/clusterdata
25. Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, Kudlur M et al. (2016) TensorFlow: a system for large-scale machine learning In Proc USENIX OSDI
26. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L et al. (2019) PyTorch: an imperative style, high-performance deep learning library In Proc. NeurIPS
27. Zhu R, Zhao K, Yang H, Lin W, Zhou C, Ai B, Zhou J et al. (2019) Aligraph: a comprehensive graph neural network platform arXiv preprint arXiv:1902.08730
28. Weng Q, Xiao W, Yu Y, Wang W, Wang C, He J, Li Y, Zhang L, Lin W, Ding Y (2022) MLaaS in the wild: workload analysis and scheduling in large-scale heterogeneous GPU clusters to appear in the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI '22), Renton, WA
29. Hastie T, Tibshirani R, Friedman J (2009) Model assessment and selection In: The elements of statistical learning Springer Series in Statistics. Springer, New York https://doi.org/10.1007/978-0-387-84858-7_7

30. Wilcoxon F (1992) Individual comparisons by ranking methods. In: Kotz S, Johnson NL (eds) Breakthroughs in statistics. Springer series in statistics, 1st edn. Springer, New York, pp 196–202