Integrating Clustering and Regression for Workload Estimation in the Cloud

Journal:	*Concurrency and Computation: Practice and Experience*
Manuscript ID	CPE-20-0281.R1
Editor Selection:	Special Issue Submission
Wiley - Manuscript type:	Special Issue Paper
Date Submitted by the Author:	04-May-2020
Complete List of Authors:	Yu, Yongjia; Changzhou College of Information Technology Jindal, vasu; University of Texas at Dallas Erik Jonsson School of Engineering and Computer Science, School of Engineering and Computer Science Yen, I-Ling; University of Texas at Dallas Erik Jonsson School of Engineering and Computer Science, School of Engineering and Computer Science Bastani, Farokh; University of Texas at Dallas Erik Jonsson School of Engineering and Computer Science, School of Engineering and Computer Science Xu, Jie; University of Leeds, School of Computing Garraghan, Peter; Lancaster University, School of Computing and Communications
Keywords:	Workload estimation, workload clustering, dynamic time warp distance, cloud computing

Integrating Clustering and Regression for Workload Estimation in the Cloud

Yongjia Yu, Vasu Jindal, I-Ling Yen, Farokh Bastani, Jie Xu, Peter Garraghan

***Abstract*—Workload prediction has been widely researched in the literature. However, existing techniques are per-job based and useful for service-like tasks whose workloads exhibit seasonality and trend. But cloud jobs have many different workload patterns and some do not exhibit recurring workload patterns. We consider job-pool based workload estimation, which analyses the characteristics of existing tasks' workloads to estimate the currently running tasks' workload. First cluster existing tasks based on their workloads. For a new task *J*, collect the initial workload of *J* and determine which cluster *J* may belong to, then use the cluster's characteristics to estimate *J*'s workload. Based on the Google dataset, the algorithm is experimentally evaluated and its effectiveness is confirmed. However, the workload patterns of some tasks do have seasonality and trend, and conventional per-job based regression methods may yield better workload prediction results. Also, in some cases, some new tasks may not follow the workload patterns of existing tasks in the pool. Thus, develop an integrated scheme which combines clustering and regression and utilize the best of them for workload prediction. Experimental study show that the combined approach can further improve the accuracy of workload prediction.**

***Index Terms*—Workload estimation, workload clustering, dynamic time warp distance, cloud computing.**

I. Introduction

Cloud computing is very popular covering e-commerce, education, government and other fields. Studies have shown significant benefits offered by IaaS cloud, including providing a greener computing environment. Many systems are developed to take advantage of the cloud, including big data analytics, offloading from mobile devices [2] and cloud storage systems [3,4,5]. However, without efficient resource management by the IaaS provider, the potential value of cloud computing cannot be fully realized. One of the important tasks in an IaaS cloud provider is to schedule resource precisely to minimize the cost of the deployment and operation of the cloud platform while fully guarantee the SLA (Service Level Agreement) for each customer. For example, reduction of operating servers by migrating tasks can save power and provide greener computation.

In order to manage cloud resources effectively, the future workloads in the cloud should be well predicted.

Many workload prediction algorithms are researched in the literature [7], [8], [9]. But they use per-job based methods (which predict a single job's future workload only based on its historical workload) and consider service-like workloads. Generally, services are accessed by users and they run year-round to process user requests. The workload characteristics of a certain systems depend heavily on its user access patterns, and mostly present periodicity and trend. Prediction can be done for these workloads using well established statistical techniques, such as autocorrelation and regression.

However, many one-time tasks in the cloud, such as animation rendering, may run for one or several days and their workloads do not have seasonality or predictable trends, such as the workload given in Fig. 1. As can be seen from Fig. 1, the irregular spikes that occur after time index 100 are difficult to model and predict using existing per-job based approaches (also validated by our experimental results). For these jobs, cloud customers are asked to set the resources requirement parameter, then the requested amount of resources are allocated for their tasks by cloud providers. Experienced cloud customers may be able to set the exact resource demands. But a lot of customers may not have sufficient experiences so that they request much more resources than what are needed. Some researches show that many customers purchase 10 times the resources than what is actually needed, and leading to low utilization of resource. In [6], the server utilization rate of real cloud computing servers were ranging from 28% to 55%, which means allocated resources were under-used. Thus, it is worthwhile to improve workload estimation methods for cloud tasks.

Cloud providers generally save job execution profiles for years. It may be assumed that the profiles of the job pool are representative. All tasks' workload pattern can be analyzed from the profiles, and can be used to match the potentially tasks submitted in the future. Based on this assumption, it is possible to calculate the workloads' pattern of "job-pool" and build the workloads model, then estimate the workload patterns of running or future tasks based on the model.

In this paper, we develop a scheme to realize the job-pool based idea discussed above work to provide better estimation of workloads that do not have service-like patterns (like Fig. 1).

Yongjia Yu is with Changzhou College of Information Technology, China, Changzhou 21300. E-mail: yuyongjia@czcit.edu.cn.

Vasu Jindal, I-Ling Yen, Farokh Bastani are with the Department of Computer Science, University of Texas at Dallas, Richardson, TX 75080. E-mail: {vasu.jindal, ilyen, Farokh.Bastani }@utdallas.edu

Jie Xu is with the School of Computing, University of Leeds, UK, Leeds, LS2 9JT. E-mail: j.xu@leeds.ac.uk.

Peter Garraghan is with School of Computing and Communications, Lancaster University, UK, Lancaster, LA1 4WA. E-mail: p.garraghan@lancaster.ac.uk.

This work is supported by Research Project of Changzhou College of Information Technology with Fund No. CXZK201704Z.

Fig.1. Sample workloads extracted from Google tracelog.

The Google dataset released in 2011 were used to explore our approach, including 12,000 servers profiles during 30 days [10]. We first cluster existing tasks and learn the statistical properties of workloads in each cluster. When a new task is submitted, we first determine its cluster based on its initial workload. Then, we use the features learned from the cluster to estimate the new job's workload. We also combine the cluster-based and regression-based workload estimation approaches to further improve the workload prediction accuracy. The major contributions of our work are as follows:

1. Per-job based workload estimation cannot work well for busty workloads with unexpected spikes and dips. Instead, we develop the job-pool based workload estimation algorithm, which uses the characteristic of the workload of the task pool to improve prediction of the workloads. We cluster historical workloads and use the cluster data to help estimate workloads of new tasks categorized into the same cluster.
2. Several cluster-based workload estimation algorithms are developed, using the statistical properties of the cluster and explored the effectiveness of these algorithms.
3. For some clusters, the job-pool based workload estimation approach may not be as effective as per-job based approach, especially for jobs with very low and constant (fluctuating around a constant) usage patterns. Also, some individual tasks do not follow the workload patterns of existing tasks in the pool, i.e., they are poorly clustered. Thus, we combine our approach with ARIMA(Autoregressive Integrated Moving Average) to improve the estimation accuracy. The combined approach works very well for almost all tasks. With our integrated approach, cloud providers can achieve accurate workload estimation even with a short initial workload information.
4. Experiments are made to calculated the effectiveness of our approach, then compare it with the requested workload as well as conventional per-job based workload estimation approaches. The results show that the job-pool based approach can improve workload prediction accuracy significantly, especially for workloads with unexpected bursts, like the one shown in Fig. 1. The combined approach makes further improvement in prediction accuracy from the cluster-based approach.

The remaining paper is organized as follows. Section 2 presents related works in the prediction and clustering of workload. Section 3 discusses our clustering approach and the details of our workload estimation algorithm. Section 4 presents the results of cluster prediction and workload estimation. Section 5 discusses how to integrate ARIMA with our approach. Section 6 make a summary of the paper and discusses the future direction.

II. RELATED WORK

A. Workload Predictions

In the literature, many workload prediction methods have been researched. Many workload prediction works consider web services systems which interacts with users, their workloads present seasonality or predictable trends. In [7], the auto-correlation function is used to get the periodic workload patterns, and the aggregate difference between each occurrence of the pattern with the actual workload is calculated, then the trending synthetic workload is generated, finally the workload placement recommendations are given accordingly. In [8], a second order ARIMA Model is used to predict the single job's future workload, then a look-ahead resource allocation algorithm is proposed based on the prediction. In [9], the Grey Forecasting Model is used, as the model can predict workload based on short historical workload and evaluate workload tendency efficiently. But this model has its own limitation. First, the prediction is coarse grained because the workloads are grouped into only four levels. Then, the prediction is based on seasonal workload so that only the same season tasks can be predicted well. In [11], a hybrid method is proposed, which combines autoregression and confidence interval estimations for long term workload predictions in grid computing environments. To eliminate the effect from noisy data, two types of filters are used before making predictions, including the Kalman filter for minimizing measurement errors and the Savitzky-Golay filter for smoothing the data. A different technique for resource demand prediction has been proposed in [12]. It first identifies historical workload patterns. Then it matches current workload patterns with these historical patterns based on the initial time series and uses the matching patterns for workload prediction. A weighted interpolation is applied to the patterns for better prediction results.

All the workload prediction approaches above are based on per-job. They need a significant amount of historical workload data, and only can predict the future workload of tasks which have same predictable trend of historical data. The string matching based method in [12] can be more sensitive to historical patterns that have few or even a single occurrence, but may be less sensitive to the frequency and timing of repetitive patterns. However, none of them is capable of predicting sudden bursts in the workload that have not appeared in historical workloads. More specifically, they are not able to predict the irregular changes shown in Fig. 1.

Some literatures predict the workload and resource management of virtual machines [13,14]. In [13], the algorithm calculates how much a given virtual machine can gain from dynamic management, and the auto regressive process forecasts the probability distribution, then the management algorithm allocates virtual machine dynamically to reduce the amount of physical capacity. In [14], the load balancer predicts the future load of servers, and migrates virtual instances to the server with the lowest load to achieve workload balancing. These works use basic workload prediction methods based on the VMs' (Virtual Machine) workload patterns with seasonality or trend, so their approaches have the same problems as

discussed above.

Another direction of prediction methods makes use of the knowledge of the applications to facilitate application-specific workload prediction. In [15], applications are first instrumented to extract the execution features and then these features are correlated with the workload patterns of the application. In [16], application-specific features, such as the type of the application, and the number of objects in a rendering application, etc., are used to predict the future workload. In [17], the support vector machine is used to predict resource demands in Software as a Service (SaaS) applications. This paper proposes to use SVM to correlate the internal features of the application to the workload patterns. The methods discussed above may be able to predict workloads effectively (including workloads presented in Fig. 1). However, they are not feasible methods for cloud environments since cloud providers will not have access to application-specific knowledge of client applications.

B. *Workload Clustering*

A large database of execution profiles is saved by cloud provide. It may be assumed that the profiles of the job pool are representative and tasks' workload pattern can be analyzed from the profiles, then can be used to match the potentially tasks submitted in the future. Some literatures cluster workload and analyze each cluster's characteristics [18, 19], but they do not use the information to improve workload prediction. In [20], we cluster the workload and calculate each cluster's average workload, then use the information to predict the workload of the new tasks belonging to the cluster. In [21, 22], we cluster the workload and build each cluster's neural network model, then use the model to predict the workload of the new tasks belonging to the cluster. In [23], the authors firstly make the clustering, then calculate the product of each cluster's submission rate and average workload, finally predict the servers' workloads. But the server information is not enough for making accurate decisions of resource management.

Fig. 2. Overview of the job-pool approach for workload estimation.

III. JOB-POOL BASED WORKLOAD ESTIMATION

In Fig. 2, the flowchart of job-pool based workload estimation approach is shown. The solid lines represent the execution flows, and the dash ones show the data flows. From the execution information of submitted tasks in the cloud environment (such as Google trace data), a large set of historical workload patterns is prepared.

First, the workload patterns of tasks in the dataset are clustered and labeled. After the clustering, N workload clusters are obtained denoted by $\{C_1,C_2,...,C_N\}$. Then compute each cluster's average workload, the medoid of cluster C_i is denoted by c_i.

When a new task J is submitted to the host, the resources requested by the cloud customer is fully allocated for J because no workload information is collected. During the execution of the task, the monitoring subsystem collect its initial workload. For each cluster C_i, the distance between its medoid c_i and the initial workload of J is computed, then which cluster(s) J's workload pattern belongs to is determined.

Based on the workload patterns in the cluster(s) which J belongs to, the future workload of J is estimated and sent to the VM placement subsystem. All tasks' workloads are checked periodically by the placement decision algorithm to decide how to adjust resource allocations for task J.

The workload of J is continuously collected, and the new estimated workload is sent to the VM placement subsystem. The estimation process of J keeps working until J terminates. Then the whole workload pattern of J is integrated into the dataset of job-pool. And J's full workload will be clustered, and the characteristics of the J's cluster will be updated. When many tasks' full workloads are integrated into the dataset, the precision of the clustering may decrease, then a re-clustering will be executed to solve problem [3]. The details of each step in our scheme are explained as follows.

A. *Notation*

First, notations used in the workload estimation scheme are listed [22].

T: The length of tasks which are used to clustering.

IT: The initial length of task.

Data obtained from the clustering algorithm:

$C=\{C_1,C_2,...,C_K\}$: The set of K workload clusters.

$c_i, 1\le i\le K$: The medoid of C_i.

$m_i, 1\le i\le K$: The number of tasks in C_i.

$C_i=\{J_{ij}\mid 1\le j\le m_i\}$: The set of tasks in C_i.

$L_{ij}(t), 1\le i\le K, 1\le j\le m_i$: The workload of task J_{ij}.

$CL_i(t), 1\le i\le K$: Average of $L_{ij}(t)$ for all tasks in C_i, where

$$CL_i(t)=\frac{\sum_{J_{ij}\ in\ C_i} L_{ij}(t)}{m_i},\ 1\le t\le T$$

$CL_{\sigma_i}(t), 1\le i\le K$: Standard deviation of $L_{ij}(t)$ for all tasks in C_i, where

$$CL_{\sigma_i}(t)=\sqrt{\frac{\sum_{J_{ij}\ in\ C_i}(L_{ij}(t)-CL_i(t))^2}{m_i}},\ 1\le t\le T$$

$ST_{ij}(t), 1\le i\le K, 1\le j\le m_i$: The stability of each workload relative to the cluster average (i.e., $L_{ij}(t)-CL_i(t)$) as a ratio to the standard deviation of the workloads in the cluster $CL_{\sigma_i}(t)$. It indicates how much fluctuation the job has relative to the average workload pattern. Formally we have

$$ST_{ij}(t)=\frac{|L_{ij}(t)-CL_i(t)|}{CL_{\sigma_i}(t)},\ 1\le t\le T$$

$ST_{ij}[P], 1\le i\le K, 1\le j\le m_i$: For each task, we compute its average stability over a time period P. P could be IT or T or any other period.

$$ST_{ij}[P]=\frac{\sum_{t=1}^{P} ST_{ij}(t)}{P}$$

$ST_{\sigma_{ij}}[P], 1\le i\le K, 1\le j\le m_i$: The standard deviation of $ST_{ij}(t)$ over a time period P.

$$ST_{\sigma ij}[P] = \sqrt{\frac{\sum_{t=1}^{P}(ST_{ij}(t) - ST_{ij}[P])^2}{P}}$$

$CD_i, 1 \le i \le K$: The average distance between medoid c_i and tasks in C_i(Note that $dist(L_{ij}, L_{c_i})$ is the function of measuring distance between the workload of task J_{ij} and medoid c_i, and is defined in section 3.2).

$$CD_i = \frac{\sum_{J_{ij}\ in\ C_i} dist(L_{ij}, L_{c_i})}{m_i * T}$$

$CD_{\sigma i}, 1 \le i \le K$: The standard deviation of distances between the medoid c_i and tasks in C_i.

$$CD_{\sigma i} = \sqrt{\frac{\sum_{J_{ij}\ in\ C_i}(\frac{dist(L_{ij}, L_{c_i})}{T} - CD_i)^2}{m_i}}$$

Note that $L_{ij}(t)$, $CL_i(t)$, $CL_{\sigma i}(t)$, $ST_{ij}(t)$ are functions over time t. Depending on where they are used, we may have $1 \le t \le T$ or $1 \le t \le IT$.

When considering the new task J, the notations related to a single task can be applied to J. For example, L_J denotes the workload of J. We use $CJ = \{CJ_1, CJ_2, ...\}$ to denote the cluster(s) a new task J may belong to (predicted). According to the notations given above, index i is used to index the K clusters. For example, $CL_i(t)$ is the average workload of cluster C_i and its complete form is CL_{C_i}. When these notations are applied to the clusters for the new task J, we need to use the full cluster notation for indexing, e.g., CL_{CJ_1}, CL_{CJ_2}, CD_{CJ_i}, etc. When performing workload estimation for the new task J, we use $PL_J(t)$ to denote the estimated workload for J to differentiate it with the actual workload $L_J(t)$ of J. Also, we use RL_J to denote the workload given by the user when submitting a task (Note that RL_J is a constant, not a function of t).

B. *Clustering the Pool of Workloads*

The workload of each task is considered as a time series and the K-medoids algorithm [24,25] is applied to cluster historical tasks' workloads. The distance measurement impacts clustering significantly. For two tasks J_i and J_j, the distance between them is the distance between workloads L_i and L_j, where L_i and L_j can be viewed as two time series. The distance between two time series can be measured in different ways, such as Euclidean distance, Manhattan distance. But as these metrics align different time series and compare the pairs of points rigidly, the distances cannot be calculated precisely when there are minor pattern shifts.

For accurate calculation of the distance between two time series, DTW (Dynamic Time Warp) distance [27,28,29] is used to measure. DTW on time series is like Levenshtein distance on string sequences. It allows deletion and insertion of data points during comparison adding penalty value to the distance. A data point of one time series is compared to the corresponding data points as well as the neighboring data points (in the worst case, all data points) in another time series, then take the minimal distance into account. The definition of DTW distance is given in the following.

Let $dtw(X,Y)$ denote the dynamic time warp distance between time series $X = (x_1, x_2...., x_n)$ with length n and time series $Y = (y_1, y_2...., y_m)$ with length m.

$$dtw((x_1, x_2...., x_n),(y_1, y_2...., y_m)) = |x_n - y_m| + min\begin{Bmatrix} dtw((x_1, x_2...., x_{n-1}),(y_1, y_2...., y_m)), \\ dtw((x_1, x_2...., x_n),(y_1, y_2...., y_{m-1})), \\ dtw((x_1, x_2...., x_{n-1}),(y_1, y_2...., y_{m-1})) \end{Bmatrix}$$

Based on the DTW distance definition, we have:

$$dist(L_i, L_j) = \frac{dtw(L_i, L_j)}{max\ (\|L_i\|, \|L_j\|)}.$$

C. *Decide the Cluster of a New Job*

For a new task J, we determine the cluster it belongs to. The detailed steps for determining CJ are given as follows:

1. Compute JD_i, the DTW distance between $L_{c_i}(t)$ and $L_J(t)$, only considering the time period IT, i,e., $JD_i = dist(L_J(t), L_{c_i}(t))$, $1 \le t \le IT$.
2. Compute the fuzzy membership value JM_i, which indicates how likely J is in cluster C_i, where
$$JM_i = \frac{CD_i + r * CD_{\sigma i}}{JD_i}$$
where r is a predetermined constant and it is set to 3 in this paper (based on the common 3σ rule).
3. Select h clusters $C_{i_1},...,C_{i_h}$ such that
$$\min_{1 \le i_j \le K} -S\ (G, JD_{i_j})$$
Assume that $\{C_{i_1},...,C_{i_h}\}$ is ordered by their JD values, i.e., C_{i_1} has the minimal DTW distance from L_J.
4. Put C_{i_1} into CJ.
5. If $JM_{i_1} < th_1$, then put C_{i_j} into CJ if $JM_{i_j} \ge th_2$.

The function $\min_{range\ of\ i} -S(x, y_i)$ used in Step 3 is an extended function of "min". It selects x items with the minimal y values from the given range of i (when $x = 1$, min-S = min). In the above algorithm, we consider two thresholds to determine how many clusters the new job J should belong to, th_1 and th_2. Maximally, h clusters may be considered for CJ. So in Step 3, we select G clusters, $\{C_{i_2},...,C_{i_h}\}$, among all K clusters, with the minimal JD values. Since C_{i_1} has the minimal DTW distance from L_J, it has to be the choice of J's cluster (Step 4). But if $JM_{i_1} < th_1$, then we believe that the likelihood of job J belonging to C_{i_1} is not sufficiently high, so we consider additional clusters from $\{C_{i_2},...,C_{i_h}\}$ as J's clusters. The threshold th_2 is used for this selection. We consider that J may also belong to C_{i_j} if $JM_{i_j} \ge th_2$, for all j, $2 \le j \le h$. Step 5 performs this selection. Assume that $CJ = \{CJ_1, CJ_2, ...\}$ is ordered by the JD value, i.e., $JD_{CJ_1} \le JD_{CJ_2} \le ....$

D. Workload Estimation

We estimate the workload of a new task J based on the statistical properties of its cluster CJ obtained in Section 3.3. We consider several workload estimation algorithms using different statistical properties of a partial set of or all of the tasks in CJ.

CAE Algorithm (Cluster Average based Estimation) uses clusters' average workload for estimation.

KNNE Algorithm (K-Nearest Neighbor based Estimation) selects K nearest tasks in the cluster(s) based on DTW distance to J, then use their average workload for estimation.

The KNST Algorithm (K-Nearest Stability Neighbor Estimation) select K nearest tasks based on the stability and distance, then use their average workload for estimation.

The OKNST Algorithm (Overestimation on K-Nearest Stability Neighbor) select K nearest tasks based on the stability and distance, then overestimate the workload based on the fluctuation of these tasks. OKNST is to give a better SLA assurance at the cost of potentially wasted resources.

For comparison with conventional algorithm, we choose ARIMA to represent the conventional per-job based workload prediction algorithms.

All algorithms are discussed in the following subsections.

1) CAE Algorithm

A simple way to estimate the workload for a new task J is to use the cluster average workload.

When $\|CJ\| = 1$, then the cluster average workload is used to be the estimated workload of J.

$$PL_J(t) = CL_{CJ_1}(t), IT < t \le T$$

When CJ includes more than one cluster, a weighted sum based on JM_i is used, where

$$PL_J(t) = \frac{\sum_{i=1}^{\|CJ\|} JM_{CJ_i} * CL_{CJ_i}(t)}{\sum_{i=1}^{\|CJ\|} JM_{CJ_i}}, IT < t \le T.$$

2) KNNE Algorithm

The workloads of a task in a large cluster could vary quite significantly. Instead of using the average workload of the entire cluster(s), we can select some tasks in the cluster(s) that are most similar to J and use their workloads to estimate J's workload. Here, we consider selecting M (M nearest neighbors, though the name is KNNE) tasks whose workloads are closest to L_J. Let $KN = \{kn_1,...,kn_M\}$ denote the M tasks from all clusters in CJ that are most similar to J in their DTW distances. Specifically, kn_i satisfies

$$\min_{i,j,\, C_i \in CJ} -S\left(M, dist(L_J(t), L_{C_{i,j}}(t))\right), 1 \le t \le IT.$$

Then, KNNE calculates the average workload of the tasks in KN as the estimated workload of J.

$$PL_J(t) = \frac{\sum_{i=1}^{M} L_{kn_i}(t)}{M}, IT < t \le T$$

3) KNST Algorithm

In KNNE algorithm, the M nearest neighbours are selected solely based on the DTW distance. Here we also take the stability (degree of fluctuation relative to cluster average) of the tasks into account when considering the M nearest neighbors.

1. Compute the stability of the new task J relative to each of the clusters in CJ, i.e., $ST_{CJ_iJ}(t), 1 \le i \le h$. Then, compute the average and standard deviation of $ST_{CJ_iJ}(t)$, over time period IT, i.e., $ST_{CJ_iJ}[IT]$, $ST\sigma_{CJ_iJ}[IT]$.
2. The KNST algorithm selects M closest tasks based on the dissimilarity metric $dissim$ defined on three attributes, the DTW distance, the average stability over IT, and the standard deviation of stability over IT. Formally,
$$dissim\left(L_J(t), L_{CJ_{i,j}}(t)\right) = dist\left(L_J(t), L_{CJ_{i,j}}(t)\right) * |ST_{CJ_iJ}[IT] - ST_{CJ_{i,j}}[IT]| * \left|ST\sigma_{CJ_iJ}[IT] - ST\sigma_{CJ_{i,j}}[IT]\right|.$$
3. Select the set of M tasks that are most similar to J based on the $dissim$ metric, and let $KN = \{kn_1,...,kn_M\}$ denote the set. Specifically kn_i satisfies
$$\min_{i,j,\, C_i \in CJ} -S\left(M, dissim(L_J(t), L_{C_{i,j}}(t))\right), 1 \le t \le IT.$$
4. The estimated workload of J is
$$PL_J(t) = \frac{\sum_{i=1}^{M} L_{kn_i}(t)}{M}, IT < t \le T.$$

4) OKNST Algorithm

The first three steps of OKNST are the same as those of KNST. But the notation used in Step 3 needs to be revised to make the rest of the computation clear. Here, we give the details of Steps 3 onwards of OKNST.

3. Let $KN_{CJ_i} = \{L_{CJ_ij_1}, L_{CJ_ij_2}, ...\}$ denote the set of tasks selected by KNST (KN_{CJ_i} is a subset of the M tasks most similar to J) and belongs to cluster CJ_i. Specifically, $L_{CJ_ij_l}$ satisfies
$$\min_{i,j,CJ_i \in CJ} -S\left(M, dissim(L_J(t), L_{CJ_{i,j}}(t))\right)$$
and $M = \sum_i \|KN_{CJ_i}\|$.
4. For each job in KN_{CJ_i}, compute its over estimated stability value, denoted as $oeST_{CJ_ij_l}$, where
$$oeST_{CJ_ij_l} = ST_{CJ_ij_l}[T] + rp * ST\sigma_{CJ_ij_l}[T].$$
Here rp is the constant controlling the degree of over-estimation.
5. Compute the estimated workload of J in cluster CJ_i, denoted as $PL_{CJ_iJ}(t)$, where
$$PL_{CJ_iJ}(t) = CL_{CJ_i}(t) + \max_{1 \le j \le \|KN_{CJ_i}\|} oeST_{CJ_ij} * CL_{\sigma CJ_i}(t), \quad IT < t \le T.$$
Essentially, the maximal fluctuation ($oeST_{CJ_ij}$) among all of the jobs in KN_{CJ_i} is used as the coefficient to overestimate the workload.
6. Finally, the maximal value of $PL_{CJ_iJ}(t)$ is used for $PL_J(t)$, for each time instance t, i.e.,
$$PL_J(t) = \max_{C_i \in CJ} \left(PL_{CJ_iJ}(t)\right), IT < t \le T.$$

5) ARIMA

To allow thorough evaluation of the job-pool based workload estimation approach, we also compare the algorithms

introduced above with conventional per-job based workload prediction algorithms. We choose to use ARIMA to represent the conventional per-job based workload prediction algorithms.

The ARIMA model consists of three components, autoregression, integration and moving average. Each component has an associated parameter to be determined from the historical data. The mathematical formulation of the $ARIMA\ (p,d,q)$ model using lag polynomials is:

$$Y_t = (1-L)^d t$$

for non-stationary time series and

$$\left(1-\sum_{i=1}^{p}\emptyset_i L^i\right)Y_t = \left(1+\sum_{i=1}^{q}\theta_i L^i\right)\varepsilon_t$$

for stationary time series, where p, d and q are natural numbers. Specifically, p is the number of autoregressive terms, d is the number of non-seasonal differences needed for stationarity, and q is the number of lagged forecast errors in the estimation.

We use the auto.arima function in forecast R package for per job workload estimation. The ARIMA function uses a variation of the Hyndman and Khandakar [30] approach. In this approach, unit root tests, minimization of the Akaike Information Criterion (AIC) and Maximum Likelihood Estimation (MLE) are used to obtain the three parameters, p,d,q , in the ARIMA model.

IV.EXPERIMENTAL RESULTS

We use the Google trace log, consisting of workload data from a real cloud environment during 29 days, to evaluate our workload estimation approaches. In Google data, each task is uniquely labeled by its Job Id and Task Index. Information of each task includes CPU, memory, and disk usages, etc. From our preliminary analysis of the trace log data, memory and disk workloads are relatively stable and easy to predict, while CPU usage patterns of many tasks are highly dynamic and the prediction can be challenging. Thus, the prediction of tasks' CPU usage of the tasks is considered. In the Google dataset, the CPU usage is recorded in every 5 minutes. Each usage data point is the average value of usages during a recording period, and has been normalized into the range of [0, 1.0] [10].

According to the others literatures, the data of day 18 is representative of the all trace log[31], therefore tasks from day 18 with 238 to 288 (24 hours) data points are selected. For the tasks with less than 288 data points, we pad them with 0s. We consider tasks with long execution time because workload estimation is most meaningful for these tasks. Among all the tasks satisfying the criteria above, we randomly choose 8639 tasks as the basis for clustering and use G_0 to denote this dataset. We then randomly select 2197 tasks from G_0 and mark this subset as G_1. Also we select another 2141 tasks and the dataset is marked as G_2. G_0 and G_2 are disjoint. We use G_1 to test the accuracy of cluster prediction. Then we perform workload estimation for tasks in G_1 and G_2 and determine the accuracy of our workload estimation approaches by comparing the estimated workload with the actual workload.

For each new task J, we measure the accuracy of its workload estimation by metrics defined in the following [22].

TCP : The cluster estimation accuracy. For a task J, if the cluster to which J belongs is in CJ, then we set $CP_J = 1$, otherwise $CP_J = 0$. Note that this test is only applied to tasks in G_1 (no data to determine to which cluster a task in G_2 should belong). The cluster estimation accuracy TCP is the average of CP_J for all $J \in G_1$.

$DP_J(A)$: The workload estimation accuracy, A is one of the workload estimation algorithms or the workload requested by the customer (RL). $PL_J(t)$ is the workload estimated for time t. T is the whole length of the workload time series, and $T = 288$ time indices (24 hours).

$$DP_J(A) = \frac{\sum_{t=IT+1}^{T}|L_J(t) - PL_J(t)|}{\sum_{t=IT+1}^{T}L_J(t)}$$

$TDP(A,G)$: The average accuracy of all tasks in the dataset G (G_1 or G_2), A is one of the workload estimation algorithms.

There are some tasks with strict performance requirements which needs sufficient resources to run smoothly. Therefore, we define under estimation statistics.

$UT_J(A) = \{t_i \mid PL_J(t_i) < L_J(t_i)\}$: is the set of time indices when the estimated workload by algorithm A is lower than the actual workload.

$$UP_J(A) = \frac{\sum_{t_i\ in\ UT_J}(L_J(t_i) - PL_J(t_i))}{\sum_{t=IT+1}^{T}L_J(t)}$$

$UP_J(A)$ can be considered as the measure of how much the workload is under estimated by algorithm A for a task J. For the test dataset, we compute $TUP_J(A,G)$, which is the average $UP_J(A)$ over all tasks in the dataset G. Note that UP_J can only be a reference metric because over estimation can yield low UP_J, but it does not imply the accuracy of the estimation. Thus, we also consider $Avg_J(A)$, which is the average of $PL_J(t)$ estimated by algorithm A, i.e., $Avg_J(A) = \frac{\sum_{t=IT+1}^{T}PL_J(t)}{T-IT}$, and $TAvg(A,G)$, which is the average $Avg_J(A)$ over all tasks in the dataset G. If two algorithms have similar $TAvg$, then TUP data can serve as an indicator about how well the algorithms can avoid under estimation.

A. Cluster the Pool of Tasks

We use K-medoids algorithm to cluster the 8639 tasks in G_0 into 40 clusters (different K values have been explored and $K = 40$ is chosen based on silhouette value [26, 32]). We select three sample clusters and show them in Fig. 3, where one cluster is shown in one diagram. Each line in each diagram represents one time series. Time index is given on the x-axis and each time index unit represents 300 seconds. The CPU usage for each task is plotted on the y-axis [21].

(a) Cluster4

(b) Cluster39

(c) Cluster18

Fig. 3. Workload patterns from four clusters. Each cluster contains many tasks with similar workload patterns.

As can be seen, tasks in Cluster4 (Fig. 3a) have very low CPU usages and fluctuate at a high frequency between the range of [0.00, 0.09]. CPU usages of tasks in Cluster39 (Fig. 3b) have an irregular fluctuation and the fluctuation frequency is lower than those in Cluster4. Although the CPU usages are mostly within [0.10, 0.25], there are low troughs at four time indices. The drops in workloads within the contiguous troughs do not show periodicity.

Compared to other clusters, Cluster18 (Fig. 3c) shows a distinct characteristic. The CPU usages of tasks in this cluster have a small fluctuation similar to other clusters in most of the time periods, but there are several very high bursts. The time indices of the bursts are quite uniform among tasks in this cluster, with one in the beginning of the execution and the others between time 125 and 200.

If a job is clustered into, for example, cluster18, based on its initial workload, and if the clustering is accurate, then the workloads of existing tasks in cluster18 can provide valuable information to help estimate the potential spikes of the current task. Each of the other clusters that are not discussed here has its own characteristics which can be helpful for predicting special workload patterns.

B. Cluster Prediction for New Tasks

We study the accuracy of cluster prediction for jobs in G_1. The configurable variables used in the cluster prediction algorithm are set as follows: The thresholds th_1 and th_2 are set to be 1.3 and 1, respectively. Essentially, we consider a task definitely belongs to cluster C_1 if its fuzzy membership value $JM_i \leq 1.3$, i.e., its DTW distance to C_1's medoid is within 1.3σ. Otherwise, threshold th_2 determines additional clusters that the new task may belong to if its DTW distance to the medoids of those clusters are within 3σ. If a task is assigned to multiple clusters, the maximum number of clusters it can be assigned to is bounded by h. We set h to 1, 2, and 3, to study its impact on cluster prediction accuracy.

TABLE 1

Table 1 shows cluster prediction accuracy for different IT and h values. With the same h, TCP increases when IT increases. From $IT = 6$ to $IT = 24$, the improvement is significant. The increase slows down after $IT = 24$. When $IT = 36$ there is a burst in TCP. Then, from $IT = 48$ to $IT = 144$, TCP increases slowly, with a 5% improvement. TCP also increases with increasing h value. This is because a task may be clustered into h clusters and as long as one of the h clusters matches, we consider it accurate. However, the impact of h is not very significant.

C. Parameter Tuning for Different Workload Estimation Algorithms

In this section, we tune the key parameters for each algorithm discussed in Section 3 to maximize their prediction accuracy. The impact of various parameters and our final parameter selections for each algorithm are discussed in each of the following subsections.

1) CAE Algorithm

Fig. 4. $TDP(CAE,G_2)/TDP(RL,G_2)\%$ of CAE algorithm

We examine the accuracy of the CAE algorithm for workload estimation. Fig. 4 shows the relative error rates, $TDP(CAE,G_2)/TDP(RL,G_2)$, with different h settings. Note that $TDP(RL) = 6054.61\%$. Fig. 5 shows the under-estimation rates for the CAE algorithm, $TUP(CAE, G_2)$. From Fig. 4, we can see that the error rates TDP for $h = 2, 3$ are slightly higher than those for $h = 1$ but the difference is more significant when $IT = 6$. This is expected because CAE does not differentiate the selected clusters (could be 1 to h clusters) when taking the average workload as the predicted workloadxi. In Fig. 5, we can see that the under estimation rates TUP for $h = 1$ is much higher (roughly 20-35% higher) than those for $h = 2, 3$. Note that $Tavg$ for different h and IT values are in a narrow range of [0.022, 0.025], indicating that the TUP comparison is illustrative. From the figures, we can see that CAE is not suitable for workload estimation when we do not have enough initial workload data.

Fig. 5. $TUP(CAE, G_2)$ of CAE algorithm

Since $h = 3$ has similar TDP values as $h = 1$ for $IT \geq 12$, but has much lower TUP, we set $h = 3$ for CAE for subsequent experiments.

2) KNNE Algorithm

Now we study the impacts of the parameter settings for the KNNE algorithms. Fig. 6 shows the relative error rates, $TDP(KNNE,G_2)/TDP(RL,G_2)$ with different M values (which controls how many nearest neighbors will be chosen). Fig. 7 shows the corresponding under-estimation rates, $TUP(KNNE, G_2)$ for different M values.

Fig. 6. $TDP(KNNE, G_2)/TDP(RL,G_2)\%$ of KNNE algorithm.

From Fig. 6, we can see that TDP drops with increasing IT and from $IT = 6$ to $IT = 12$ there is the sharpest drop. One exception is when $M = 1$ and $IT = 24$, where TDP has the lowest value. From deep analysis, we found that there are several tasks with very high DP values (> 10000 for $IT = 6, 12, 36, 48$), which significantly raises the overall error rate TDP. The high DP values are due to the very low CPU usage (less than 0.001). When $M = 1$, only one task is chosen as the basis for workload estimation. Thus, the better choice made for $IT = 24$ leads to a lower error rate than the other cases. In terms of M, $M = 20$ has the lowest TDP for almost all

IT values.

Fig. 7. $TUP(KNNE, G_2)$ of KNNE algorithm.

Since $TAvg$ values for all M and IT settings are similar and are in the range of [0.022, 0.025], we can compare under estimation rates directly. Fig. 7 shows TUP values under different M settings for the KNNE algorithm. Obviously M=1 would not result in good estimation. But as M increases from 10 to 40, TUP also increases.

A low M value does not provide a sufficient basis for estimation and a high M value implies that the large sample space may turn out to contain outliers and impacts the estimation accuracy. Thus, we choose $M = 20$ for KNNE because it yields the best TDP and second lowest TUP.

3) KNST Algorithm

Fig. 8. $TDP(KNST, G_2)/TDP(RL,G_2)\%$ of KNST algorithm.

Fig. 9. $TUP(KNST, G_2)$ of KNST algorithm.

M is also the key parameter in the KNST algorithm. Fig. 8 and Fig. 9 show its relative error rate $TDP(KNST,G_2)/TDP(RL,G_2)$ and TUP, respectively, for different M.

Comparing Fig. 8 and Fig. 9 with Fig. 6 and Fig. 7, we can see that KNST has a similar trend as KNNE. For example, when $M = 20$, it has the lowest TDP value. But when $M = 40$, it has the lowest TUP value. $TAvg$ of KNST also has a narrow range of [0.023, 0.025], indicating the validity of the TUP based comparisons. A different observation in KNST is that TUP values do not change much from $IT = 12$ to $IT = 48$. Based on the exploration, we also choose $M = 20$ for the KNST algorithm.

4) OKNST Algorithm

OKNST algorithm targets to overestimate workloads and the RP parameter controls the overestimation degree. The relative error rates $TDP(OKNST,G_2)/TDP(RL,G_2)$ and the underestimation rates TUP for the OKNST algorithm with different RP values are shown in Fig. 10 and Fig. 11, respectively. Since $TAvg$ values for different settings vary relatively significantly, we also show them in Table 2.

Fig. 10. $TDP(OKNST, G_2)/TDP(RL,G_2)\%$ of OKNST.

Fig. 11. $TUP(OKNST, G_2)$ of OKNST.

From the figures, we can see that different RP values yield quite different TDP and TUP values. As expected, higher RP leads to higher TDP but lower TUP values. The reason for lower TUP can also be seen from $TAvg$ shown in Table 2. With increasing RP, $TAvg$ increases, which contributes to the dropping of TUP values. We choose $RP = 2$ for OKNST in the subsequent experiments.

TABLE 2

5) ARIMA

We use a variation of the Hyndman and Khandakar [30] approach (in auto.arima of R) to obtain the three parameters of ARIMA, p, d and q. The steps for building the ARIMA model are summarized as follows:

1. The number of non-seasonal differences d is determined using repeated Kwiatkowski–Phillips–Schmidt–Shin (KPSS) tests.
2. The values of p and q are then chosen by minimizing the Akaike information criterion (AIC) after applying d to the data (differencing the data d times).
   (a) The best model (with smallest AICs) is selected from the following four tests: $ARIMA$ (2,d,2) $ARIMA$(0,d,0), $ARIMA$(1,d,0), $ARIMA$(0,d,1). The four choices are selected based on the recommendation discussed in [30].
   (b) With the decision of d, we further change p and/or q from the current model by ±1 and select the better p and q values.
   (c) Repeat Step 2 (b) until no lower AICs can be found.

For each time series, a different set of p, d and q settings are selected. For example, $ARIMA$(1,1,0) is the optimum model for 29.6% of the tasks and $ARIMA$(0,1,0) is the optimum model for 23.4% of tasks in our dataset.

D. Comparison of Workload Estimation Algorithms

In this section, we compare the effectiveness of different workload estimation algorithms, including CAE, KNNE, KNST, OKNST and ARIMA. The parameter settings for each algorithm have been discussed in Section 4.3. For ARIMA, we consider its prediction at different IT values to see its prediction accuracy at early time. We also consider its continuous prediction (ARIMA-C), in which ARIMA estimates the workload of the next 12 time units at IT = 6, 12, 24, 36, …, 276. We choose to predict for 12 time units ahead because it is a reasonable time period for taking resource reallocation actions.

Fig. 12. $TDP(A,G_2)/TDP(RL,G_2)\%$ of all algorithms. OKNST uses a different scale from the other algorithms (its scale is marked by the right side vertical axis).

Fig. 13. $TUP(A,G_2)$ of all algorithms.

Fig. 12 and Fig. 13 compare the relative error rates, $TDP(A,G_2)/TDP(RL,G_2)$ and under estimation rates $TUP(A,G_2)$ of different algorithms with different IT values. Table 3 shows the $TAvg$ value of these algorithms.

It is obvious that OKNST has the highest TDP values and lowest TUP values than other algorithms due to its intentional over-estimation. The over-estimation property in OKNST can also be observed from the higher $TAvg$. However, even with over-estimation, its TDP is still only 40% of the user given workload RL and its TUP is less than 10% of RL. Thus, OKNST is a good solution if we want to have an improved resource utilization with a high SLA assurance.

TABLE 3

Among other algorithms, CAE has a higher TDP than KNNE and KNST. This is because CAE simply averages the workloads of all the tasks in the clusters for workload estimation. Its TUP is the highest among all algorithms showing that such estimation is too rough. Comparing KNST and KNNE, KNNE has a lower TDP but a slightly higher TUP than KNST. TDP of ARIMA is lower than that of CAE but higher than those of KNNE and KNST. Also, TDP of ARIMA has a burst at $IT = 12$. We found that this is because one task has a very low CPU usage (< 0.00005), and a relatively high estimation leads to an extremely high TDP (> 600000). ARIMA-C has a lower TDP than ARIMA, and it is slightly higher than KNST. But ARIMA and ARIMA-C both have lower TUP values than KNNE and KNST. This is because their workload estimates tend to be higher, as can be seen from Table 3, $TAvg$ of ARIMA is even higher than OKNST and $TAvg$ of ARIMA-C is approximately 1.5 times of $TAvg$ of KNNE and KNST.

Comparing with RL, TDP of CAE, KNNE, KNST, and ARIMA are less than 10% of $TDP(RL,G_2)$. It means that instead of simply letting users estimate resource demands of their tasks, our algorithms can be used for workload estimation to achieve much better resource utilization.

From earlier results, we can see that some algorithms may give a few estimations with very high TDP values. Thus, we construct the histograms to see the error rate distributions of the algorithms and the results are shown in Fig. 14 for $IT = 12, 48$.

From the two charts, we can see that from $IT = 12$ the error rate distributions for all algorithms shift toward the low rate side. Error rates by OKNST have a relatively even distribution over all the ranges. CAE also has quite a high count in the high TDP ranges. KNNE and KNST have the highest counts in the lowest TDP range and has very low counts in the highest TDP ranges. TDP distributions of ARIMA and ARIMA-C are better than that of CAE but worse than those of KNNE and KNST.

Now consider the scaled-up high error rate region (last two columns in the charts). ARIMA and ARIMA-C have higher counts in high error rate ranges compared to KNNE. In the 1000-10000 range, ARIMA-C has higher counts than KNNE does in both charts. Both ARIMA-C and KNNE have 0 task in the >10000 range.

Taking all factors into account, KNNE is the best workload estimation algorithm among those considered in the experimental study. It has the lowest error rate and can estimate the work load accurately at an early stage of job execution, which can greatly benefit resource allocation planning and avoid unnecessary task migration. Also, as shown in the task count analysis, per-job based workload estimation has a higher potential of making bad estimations. Thus, we can confirm that clustering does help achieve better workload estimation.

(a) IT=12

(b) IT=48

Fig. 14. Task count in each TDP range.

(1000-10000)*: Task Count is multiplied by 10.

(>10000)*: Task Count is multiplied by 100.

E. Workload Estimation for Sample Tasks

We choose two tasks, one from Cluster 1 in which the tasks have steady fluctuations around a relatively constant workload, and the other from Cluster 28, in which each task has a sharp burst roughly between time indices 220 and 230. Fig. 15 shows the estimated workload for these two tasks by various algorithms. Besides ARIMA-C, which is a continuous estimation, all other algorithms (including ARIMA) consider $IT = 24$. Note that L is the actual workload of the task as defined in Section 3.1.

From Fig. 15a, we can see that the user gives a very high workload estimate (RL) which can result in resource underutilization. OKNST over-estimates the workload. It does not show under-estimation at any time, and its estimation is much lower than that of RL. The estimation by CAE is a much lower than OKNST, but higher than other algorithms. In Fig. 15b, we scale up the estimates by other algorithms to better observe them. As can be seen from the figure, the actual workload fluctuates between [0.0028, 0.0042], which shows that Task1 steadily has a very low CPU usage. Estimation by KNST has more fluctuations than the actual workload. KNNE has the closest estimate for the workload. ARIMA cannot make long term predictions and, thus, it gives a steady estimation at around 0.0040 after time index 30. ARIMA-C tends to make predictions based on current trends and, hence, has significant deviations in the estimated workload.

(a) Task1

(b) Task 1 (scaled up for KNNE, KNST, ARIMA)

(c) Task 2

Fig. 15. Request workload, actual workload and estimation workload on single task.

From Fig. 15c, we can see that the actual workload of Task 2 (between time indices 150 and 288) fluctuates around 0.10 till the sudden burst to 0.40 at time index 225. The user estimated

workload RL is 0.3125, which is higher than the actual workload during most of the time, but lower than the peak of the burst. This implies that RL may cause resource waste most of the time but may fail to provide enough resources during the burst. OKNST is designed to over-estimate to assure the provision of sufficient resources. It has a relatively high overestimation, including the burst. It does not show under-estimation at any time. Also, OKNST gives much better workload estimation than RL.

Since we have already observed workload estimations by various algorithms for steadily fluctuating patterns in Task 1, here we focus on the estimations for the spike in Task 2 by various methods (other than RL). As expected, ARIMA is not able to make long term predictions, so it cannot anticipate any spikes or dips in the future. ARIMA-C catches up with the burst way after it happens and it delays the peak prediction till the actual peak workload period is almost over. This is expected because regression based prediction algorithms cannot anticipate unexpected changes. In contrast, CAE, KNNE, KNST and OKNST all can catch the burst on time, but they did not estimate the magnitude of the burst well. OKNST, as expected, over-estimates and CAE, KNNE, and KNST under-estimate the magnitude of the burst. Among them, KNST got the closest estimate and KNNE is almost the same as KNST, but CAE gives much lower estimate compared to the original workload.

V. COMBINING CLUSTERING AND REGRESSION FOR IMPROVED WORKLOAD ESTIMATION

From the comparison of workload estimation algorithms given in Section 4.4 and from the specific evaluation for the two sample tasks given in Section 4.5 we can see that the conventional per-job based prediction algorithm ARIMA works well for some types of tasks. From our deep analysis, when the workload of a task has steady fluctuations or steady trends, it can be estimated accurately by conventional per-job based algorithms (like ARMIA-C). On the other hand, clustering based estimation is suitable for workloads with non-smooth bursts which cannot be estimated by regression based solutions. Based on these observations, we consider a simple combination of our workload estimation approach with ARIMA-C to see whether the combined algorithm can achieve a better workload estimation. In Section 5.1, we introduce the combined algorithm. Section 5.2 analyzes the performance of KNNE and ARIMA-C for each cluster. In Section 5.3, the error rate for workload estimation by the combined workload estimation algorithm is evaluated and compared with its parent algorithms KNNE and ARIMA-C.

A. Combining ARIMA and Clustering

We design the combined algorithm "COMBINE" and consider the combination of KNNE and ARIMA at both the cluster level and the individual task level.

At the cluster level, we select the best workload prediction algorithm for each cluster based on historical data. From the training set, we decide whether the workloads in a cluster can be estimated more accurately by ARIMA-C or by KNNE. Then, the more accurate approach is used as the workload estimation method for the cluster. The combined solution at the cluster level is given in the following.

1. After clustering, consider each cluster C_i. For each workload L_{ij} in C_i, use various algorithms A_l, for all l to estimate the workload and the estimated workloads are denoted as $PL_{ij}(t, A_l)$, $IT < t \leq T$, respectively.
2. For each workload L_{ij} and its estimations $PL_{ij}(t, A_l)$, $IT < t \leq T$, compute the accuracy of the estimations. Let $AM(L_{ij}, A_l)$ denote the accuracy of the estimated workload $PL_{ij}(t, A_l)$, $IT < t \leq T$, using algorithm A_l and accuracy metric AM.
3. For each cluster C_i, compute the average accuracy (AM) of all the workloads in the cluster for each A_l, where:
$$AM(C_i, A_l) = \frac{\sum_j AM(L_{ij}, A_l)}{\|C_i\|}.$$
4. Choose the workload estimation algorithm $A(C_i)$ for cluster C_i,$1 \leq i \leq K$, where
$$A(C_i) = \max_l AM(C_i, A_l).$$
5. When a new task J arrives, decide the cluster(s) for J, namely, CJ. Note that CJ_1 is the most likely cluster for J. We use $A(CJ_1)$ to estimate the workload of J.

Fig. 16. Flowchart of the COMBINE algorithm at the task level.

At the individual task level, we consider the case when the cluster of a task cannot be determined with a high level of confidence. We will be conservative in this case and simply use the conventional workload prediction method (here it is ARIMA). Thus, a part of the flowchart given in Fig. 2 is modified and the modification is shown in Fig. 16. If $JM_{i_1} < th_1$, we use KNNE to predict J's workload. Otherwise, J cannot be well classified and ARIMA is used for its workload estimation.

B. Cluster Analysis

From the analysis given in Section 4.5, it can be seen that ARIMA-C (here, we only consider ARIMA-C) cannot predict well when the workload changes significantly and suddenly, but it performs better when the workload is stable throughout. Thus, we compare the performance of these two algorithms for individual clusters on dataset G_1 to train the algorithm selection in the combined algorithm. Here we only consider the TDP error. Fig. 17 compares TDP values of ARIMA-C and KNNE ($IT = 24$).

Fig. 17. TDP of KNNE and ARIMA-C for each clusters.

From the figure, we can see that ARIMA-C is better than KNNE in Clusters 13, 15, 38 and 39. But in other clusters, ARIMA has worse accuracy than KNNE does. In fact, in

Clusters 9, 11, 16 and 18, ARIMA-C has much higher TDP values than KNNE does because those clusters have very bursty workload patterns. Thus, we choose ARMA-C for workload estimation for Clusters 13, 15, 38 and 39 and use KNNE for the other clusters.

C. Evaluation of the Combined Algorithm

Now we compare COMBINE with ARIMA-C and KNNE ($IT = 24$) for dataset G_2. We compare the relative TDP errors of the COMBINE algorithm to its parent algorithms.

$$\frac{TDP(COMBINE,G_2)}{TDP(KNNE,G_2)} = 99.32\%$$

$$\frac{TDP(COMBINE,G_2)}{TDP(ARIMA-C,G_2)} = 63.42\%$$

The results show that COMBINE yields better workload estimation than both KNNE and ARIMA-C do. Though COMBINE does not improve significantly from KNNE, it does offer better prediction in a few clusters. Among all clusters, Clusters 13 and 15 get the best improvement and the improvements by COMBINE from KNNE in the two clusters are shown as follows.

$$\frac{TDP(COMBINE, C_{13})}{TDP(KNNE, C_{13})} = 67.05\%$$

$$\frac{TDP(COMBINE, C_{15})}{TDP(KNNE, C_{15})} = 77.35\%$$

COMBINE works better because it makes use of the advantages of different algorithms on different type of workload. For a new task which is predicted as the stable workload type, ARIMA-C is used to estimate. While the new task is predicted to the type may change suddenly, KNNE is adapted. Overall COMBINE can improve the accuracy of estimation. When more algorithms are put into the algorithm set of COMBINE, there is still improving space.

VI. Conclusion

We have developed new workload prediction algorithms to estimate the potential resource demands of tasks. We propose the job-pool based prediction approach, which generalizes the historical workloads of tasks to predict the workloads of new tasks. To realize the approach, we first cluster the workloads of existing jobs. For a new job, we predict the cluster(s) to which it may belong based on its initial workload. Then, we use the statistical workload of the cluster(s) to help estimate the workload of the new job. Experimental results show that our model is capable of making good workload estimations at an early stage of job execution, especially for jobs that have sudden and significant workload changes. Based on our analysis of Google dataset, many tasks have unexpected spikes and dips in their workload patterns. Thus, the approach we have developed can be very useful, can greatly improve workload estimation accuracy and, hence, improve resource utilization and saving power in datacenters.

References

[1] Future J. Hamilton. “Internet scale service efficiency.” Large-Scale Distributed Systems and Middleware Workshop, 2008
[2] Y. Ye, L. Xiao, I-L. Yen, F.B. Bastani. “Leveraging service clouds for power and QoS management for mobile devices.” IEEE CLOUD, Washington DC, July 2011, pp. 235-242.
[3] K. Shvachko, Hairong Kuang, S. Radia, R. Chansler, “The Hadoop distributed file system.” ACM Symposium on Mass Storage Systems and Technologies, May 2010, pp. 1-10.
[4] A. Lakshman, P. Malik. “Cassandra: A decentralized structured storage system.” ACM SIGOPS Operating Systems Review, vol. 44, No. 2, 2010, pp. 35-40.
[5] Y. Ye, L. Xiao, I-L. Yen, F.B. Bastani. “Secure, dependable, and high performance cloud storage”, SRDS, 2010, pp. 194-203.
[6] P. Garraghan, P. Townend, J. Xu. “An analysis of the server characteristics and resource utilization in Google Cloud.” IEEE Intl. Conference on Cloud Engineering, 2013, pp. 124-131.
[7] D. Gmach, J. Rolia, L. Cherkasova, A. Kemper. “Workload analysis and demand prediction of enterprise data center applications.” IEEE International Symposium on Workload Characterization, 2007, pp. 171-180.
[8] N. Roy, A. Dubey, and A. Gokhale, “Efficient autoscaling in the cloud using predictive models for workload forecasting.” CLOUD 2011, pp. 500–507.
[9] J. Jheng, F. Tseng, H. Chao, Li-Der Chou. ”A novel VM workload prediction using grey forecasting model in cloud data center“. Intl. Conference on Information Networking, 2014, pp. 40-45.
[10] C. Reiss, J. Wilkes. “Google cluster-usage traces: format + schema.” Version 2013.05.06, Google Inc.
[11] Y. Wu, Y. Yuan, G. Yang, and W. Zheng, “Load prediction using hybrid model for computational grid.” GRID 2007, pp. 235–242.
[12] E. Caron, F. Desprez, and A. Muresan, “Forecasting on grid and cloud computing on-demand resources based on pattern matching.” CloudCom 2010, pp. 456-463.
[13] N. Bobroff, A. Kochut, and K. Beaty. “Dynamic placement of virtual machines for managing SLA violations.” IFIP/IEEE Integrated Network Management, 2007.
[14] S. Daniel, M. Kwon. “Prediction-based virtual instance migration for balanced workload in the cloud datacenters.” RIT 2011.
[15] N. Doulamis, A. Doulamis, A. Litke, A. Panagakis, T. Varvarigou, E. Varvarigos. “Adjusted fair scheduling and non-linear workload prediction for QoS guarantees in grid computing.” Computer Communications, Feb. 2007, pp. 499-515.
[16] K. Dolkas, D. Kyriazis, A. Menychtas, T. Varvarigou. “e-Business applications on the Grid: A toolkit for centralized workload prediction and access.” Concurrency and Computation: Practice and Experience, April 2007, pp. 867-883.
[17] O. Niehorster, A. Krieger, J. Simon, and A. Brinkmann, “Autonomic resource management with support vector machines.” GRID 2011, pp. 157-164.
[18] Y. Chen, A. S. Ganapathi, R. Griffith, and R. H. Katz, “Analysis and lessons from a publicly available Google cluster trace.” Tech. Rep. UCB/EECS-2010-95, 2010.
[19] S. Di, D. Kondo, and F. Cappello, “Characterizing and modeling cloud applications/jobs on a Google data center.” The Journal of Supercomputing, April 2014, Vol. 96, No. 1, pp. 139– 160.
[20] J. Patel, V. Jindal, I. L. Yen, F. Bastani, J. Xu, P. Garraghan. “Workload estimation for improving resource management decisions in the cloud.” ISADS 2015, pp. 25-32.
[21] Y. Yu, V. Jindal, I-L. Yen, F.B. Bastani. “ Integrating Clustering and Learning for Improved Workload Prediction in the Cloud. ” , IEEE CLOUD, San Francisco, July 2016, pp. 876-879.
[22] Y. Yu, V. Jindal, F.B. Bastani, F. Li, I-L. Yen. “Improving the Smartness of Cloud Management via Machine Learning Based Workload Prediction", IEEE COMPSAC, Tokyo, July 2018, pp. 38-44.
[23] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, “Energy-efficient resource allocation and provisioning framework for cloud data centers,” IEEE TNSM, Vol. 12, No. 3, Sept 2015, pp. 377–391.
[24] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, A.Y. Wu. (2002). “An efficient k-means clustering algorithm: Analysis and implementation.” IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 24, No. 7, pp. 881-892.

[25] L. Kaufman and P.J Rousseeuw. “Clustering by means of Medoids. Statistical Data Analysis Based on the L_1–Norm and Related Methods”, edited by Y. Dodge, 1987, North-Holland, pp. 405–416.
[26] Z. Fang and X. Lei. Prediction of miRNA-circRNA Associations Based on k-NN Multi-Label with Random Walk Restart on a Heterogeneous Network. BIG DATA MINING AND ANALYTICS. Vol. 2, No. 4, Dec 2019, pp. 261–272.
[27] S. Salvador and P. Chan. “FastDTW: Toward accurate dynamic time warping in linear time and space. KDD Workshop on Mining Temporal and Sequential Data, 2004, pp. 70-80.
[28] H. Zhu, Z. Gu, H. Zhao, K. Chen, C. Li, L. He. “Developing a Pattern Discovery Method in Time Series Data and Its GPU Acceleration”. BIG DATA MINING AND ANALYTICS. Vol. 1, No. 4, Dec 2018, pp. 266–283.
[29] Y. Chen, Y. Zhang, J. Hu. “Multi-Dimensional traffic flow time series analysis with self-organizing maps”. Tsinghua Science and Technology. 2008, Vol. 13, No.2, pp. 220-228.
[30] J. Hyndman, Y. Khandakar. "Automatic Time Series Forecasting: The forecast Package for R". Journal of Statistical Software, 2008.
[31] I.S. Moreno, P. Garraghan, P. Townend, and J. Xu. “An approach for characterizing workloads in Google Cloud to derive realistic resource utilization models.” SOSE. 2013, pp. 49-60.
[32] R.C. de Amorim, C. Hennig. “Recovering the number of clusters in data sets with noise features using feature rescaling factors ” . Information Sciences. 2015, pp. 126-145.

TABLE 1
TCP ON DIFFERENT IT AND h VALUES

IT	$h=1$	$h=2$	$h=3$
6	67.87%	68.96%	70.10%
12	74.65%	77.97%	78.74%
24	79.06%	83.52%	84.16%
36	82.48%	85.03%	85.98%
48	81.02%	83.34%	83.25%
72	81.84%	84.30%	84.57%
96	82.66%	84.93%	85.30%
120	83.48%	85.57%	85.80%
144	86.44%	87.94%	88.21%

TABLE 2
$TAvg(OKNST,G_2)$ OF OKNST ALGORITHM

IT	RP = 0	RP = 0.5	RP = 1	RP = 2	RP = 3
6	0.03907	0.04389	0.04900	0.05925	0.06973
12	0.03650	0.04103	0.04582	0.05552	0.06549
24	0.03628	0.04079	0.04544	0.05493	0.06465
36	0.03616	0.04055	0.04506	0.05429	0.06373
48	0.03599	0.04000	0.04450	0.05326	0.06213

TABLE 3
$TAvg(A,G_2)$ ON DIFFERENT IT

IT	CAE	$KNNE$	$KNST$	$OKNST$	$ARIMA$	$ARIMA\_C$
6	0.0293	0.0241	0.0245	0.0592	0.0603	0.0333
12	0.0241	0.0227	0.0235	0.0555	0.0641	0.0333
24	0.0240	0.0226	0.0235	0.0549	0.0494	0.0333
36	0.0239	0.0228	0.0235	0.0542	0.0564	0.0333
48	0.0242	0.0228	0.0234	0.0532	0.0522	0.0333

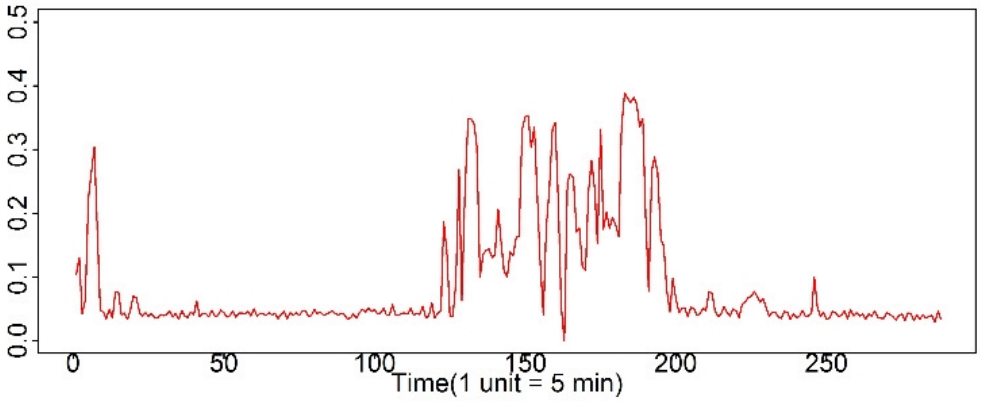


Fig.1. Sample workloads extracted from Google tracelog.

85x37mm (220 x 220 DPI)

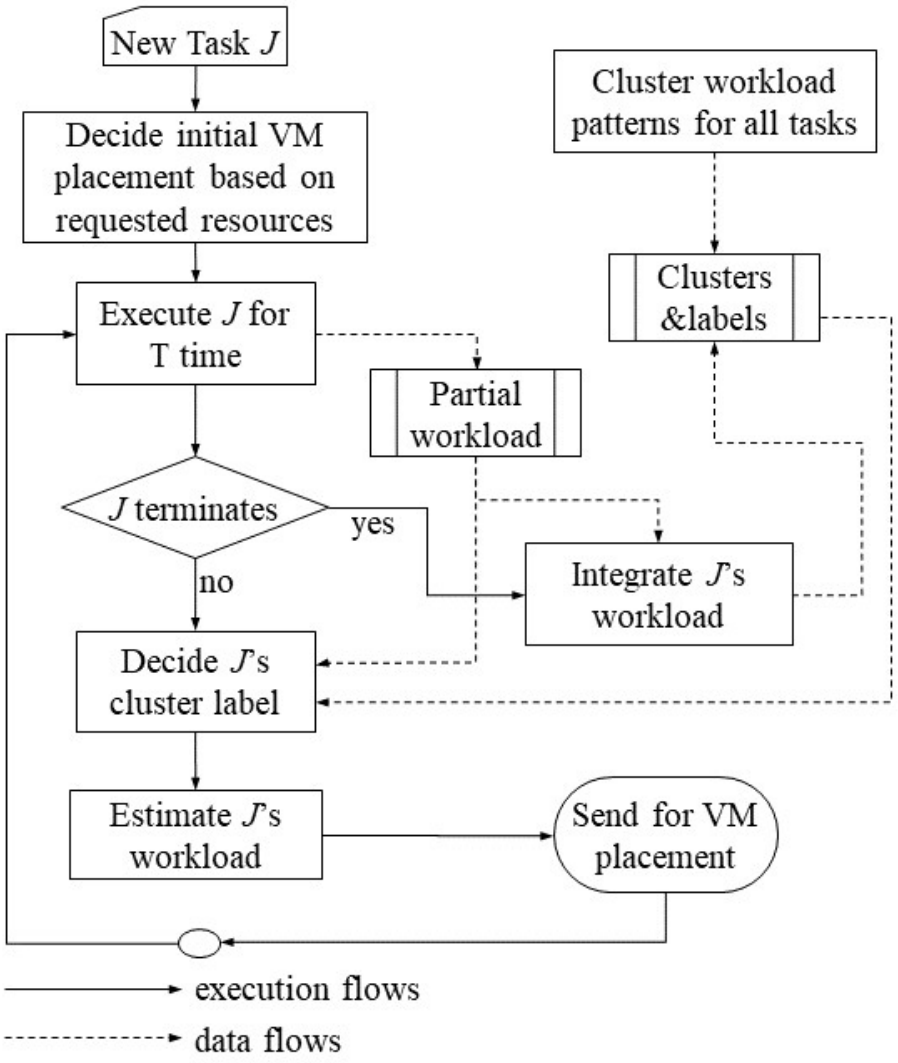
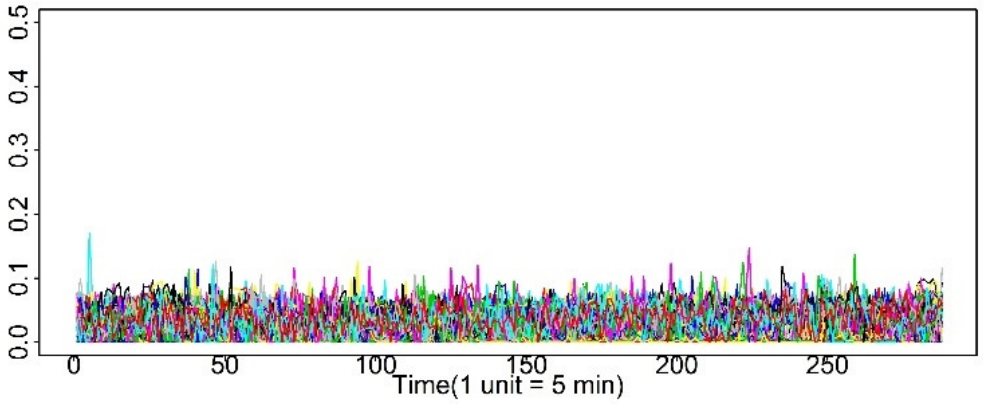


Fig. 2. Overview of the job-pool approach for workload estimation.

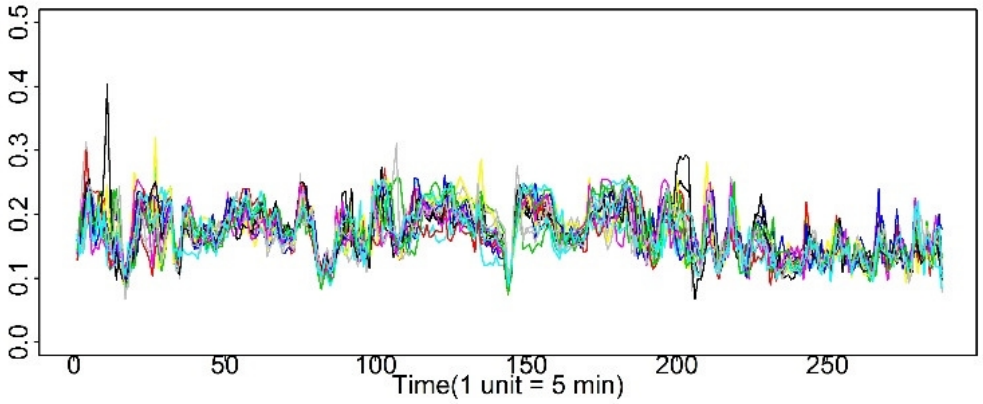
158x175mm (96 x 96 DPI)



(a) Cluster4

Fig. 3. Workload patterns from four clusters. Each cluster contains many tasks with similar workload patterns.

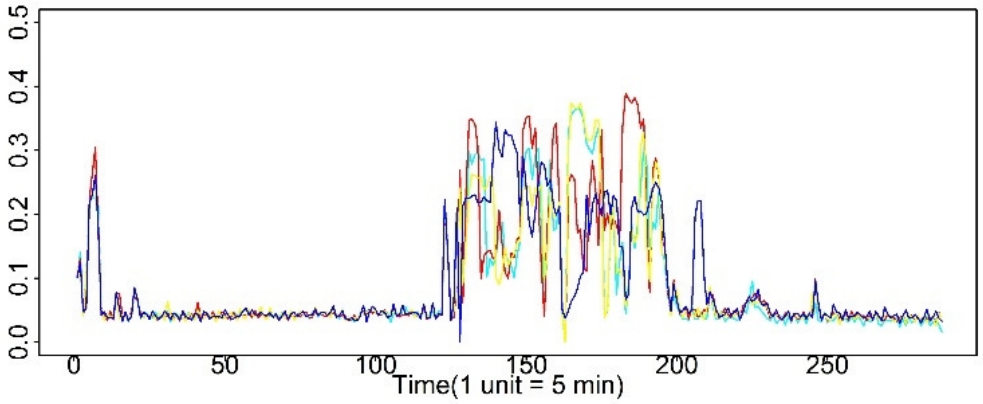
86x37mm (220 x 220 DPI)



(b) Cluster39

Fig. 3. Workload patterns from four clusters. Each cluster contains many tasks with similar workload patterns.

86x37mm (220 x 220 DPI)



(c) Cluster18

Fig. 3. Workload patterns from four clusters. Each cluster contains many tasks with similar workload patterns.

86x37mm (220 x 220 DPI)

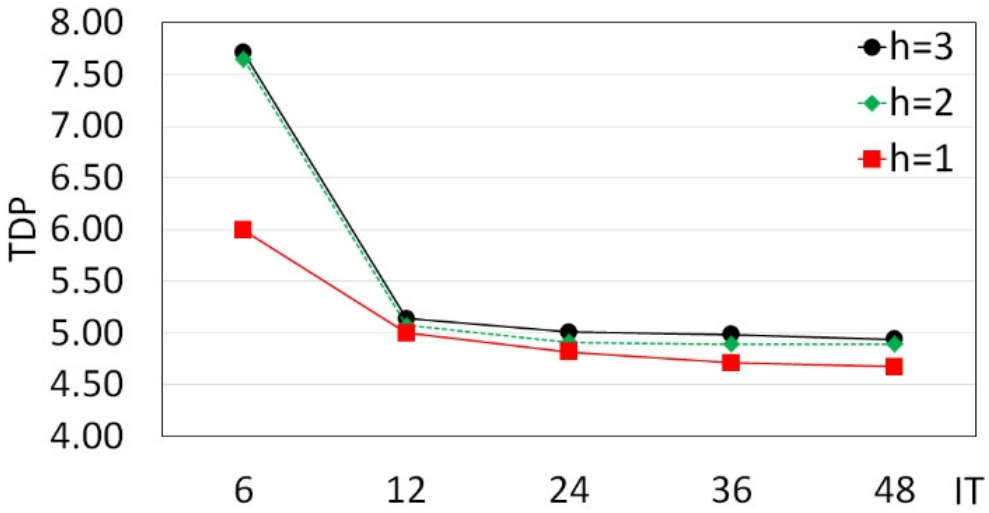


Fig. 4. *TDP(CAE,G_2)/TDP(RL,G_2)*% of CAE algorithm.

82x43mm (220 x 220 DPI)

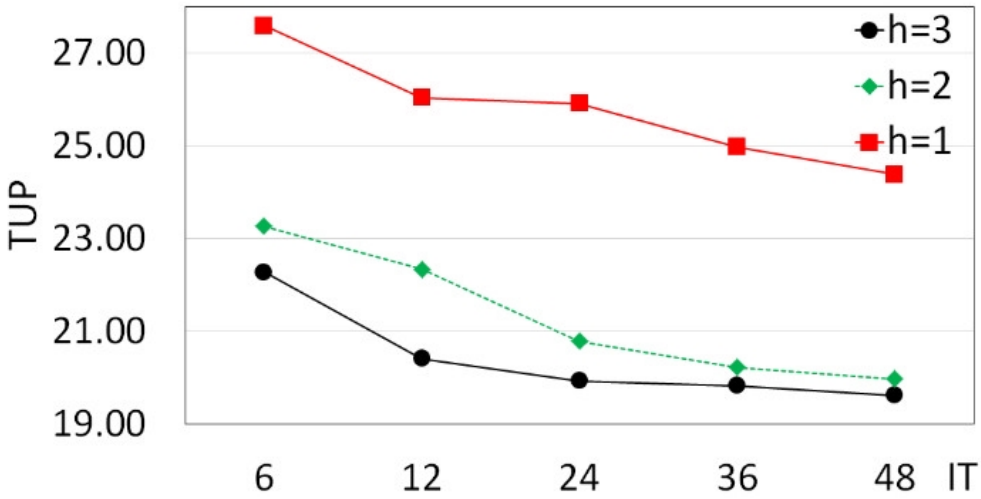


Fig. 5. *TUP(CAE,G_2)* of CAE algorithm.

81x42mm (220 x 220 DPI)

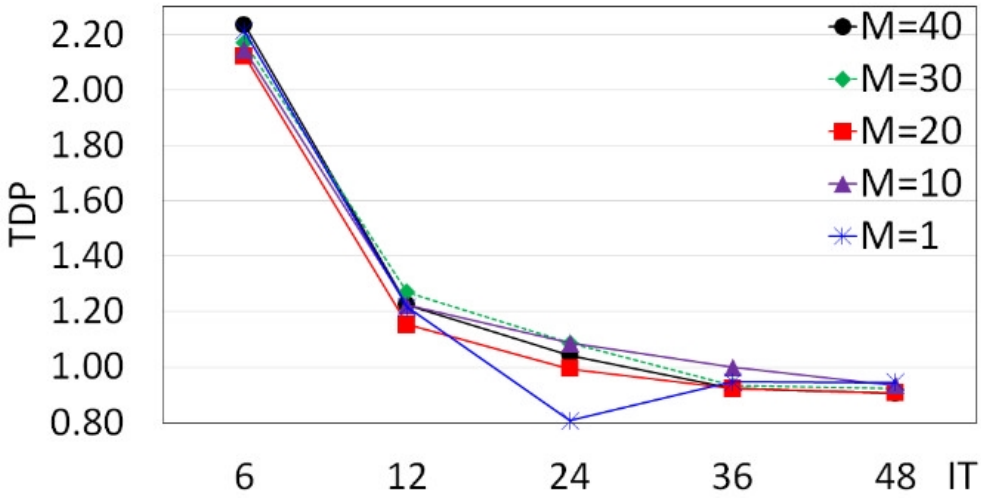


Fig. 6. *TDP(KNNE,G_2)/TDP(RL,G_2)*% of KNNE algorithm.

77x40mm (220 x 220 DPI)

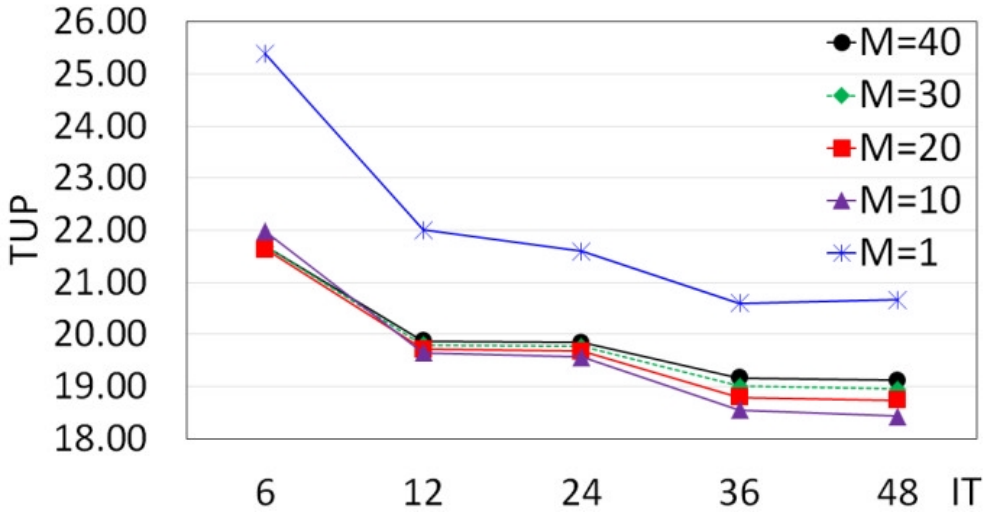


Fig. 7. *TUP(KNNE,G_2)* of KNNE algorithm.

79x41mm (220 x 220 DPI)

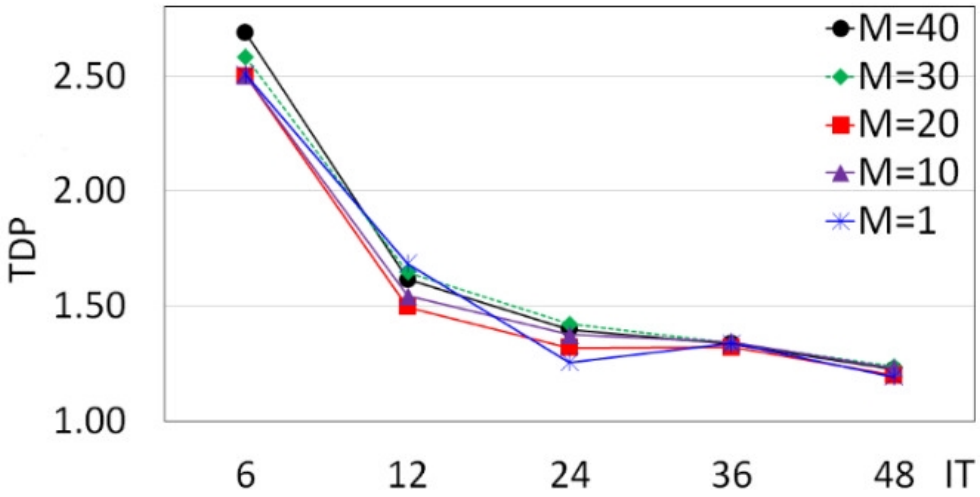


Fig. 8. *TDP(KNST,G_2)/TDP(RL,G_2)*% of KNST algorithm.

77x40mm (220 x 220 DPI)

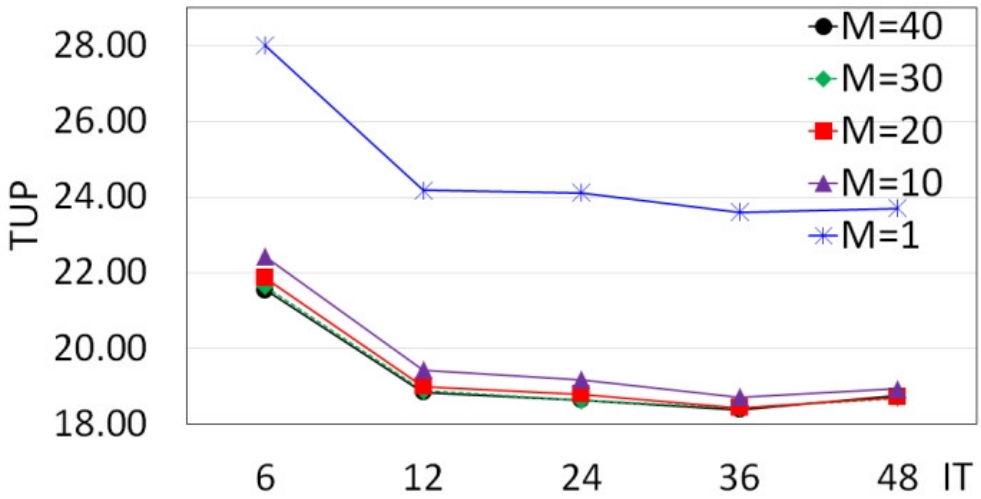


Fig. 9. *TUP(KNST,G_2)* of KNST algorithm.

80x41mm (220 x 220 DPI)

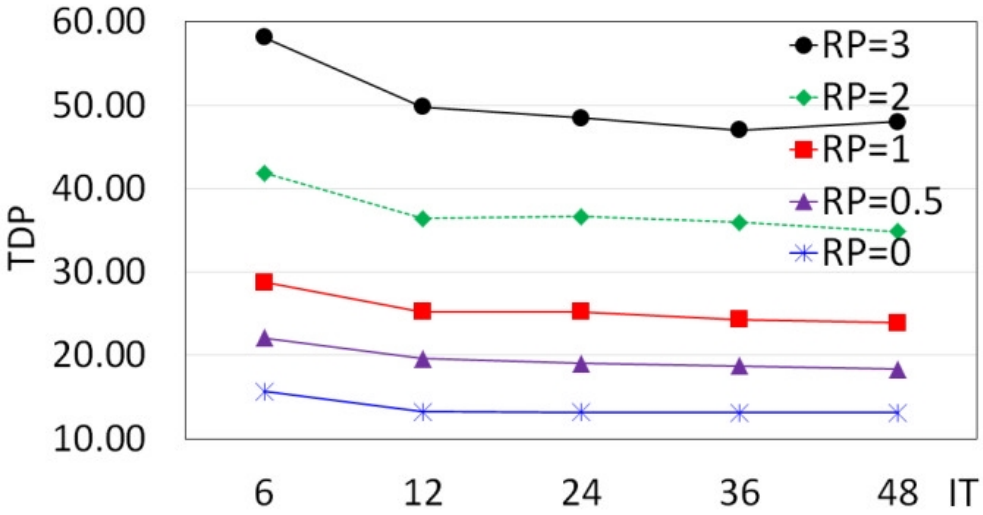


Fig. 10. *TDP(OKNST,G_2)/TDP(RL,G_2)*% of OKNST.

78x40mm (220 x 220 DPI)

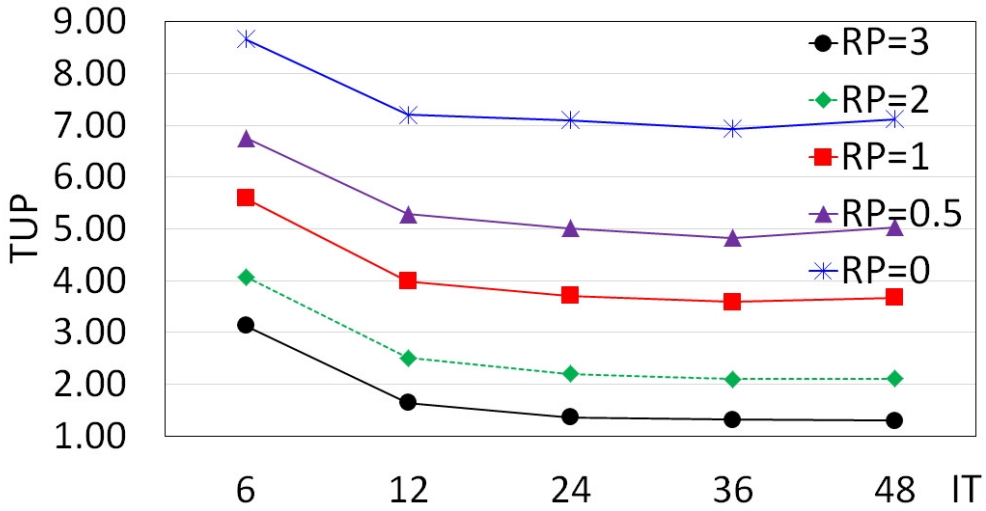


Fig. 11. *TUP(OKNST,G_2)* of OKNST.

274x142mm (96 x 96 DPI)

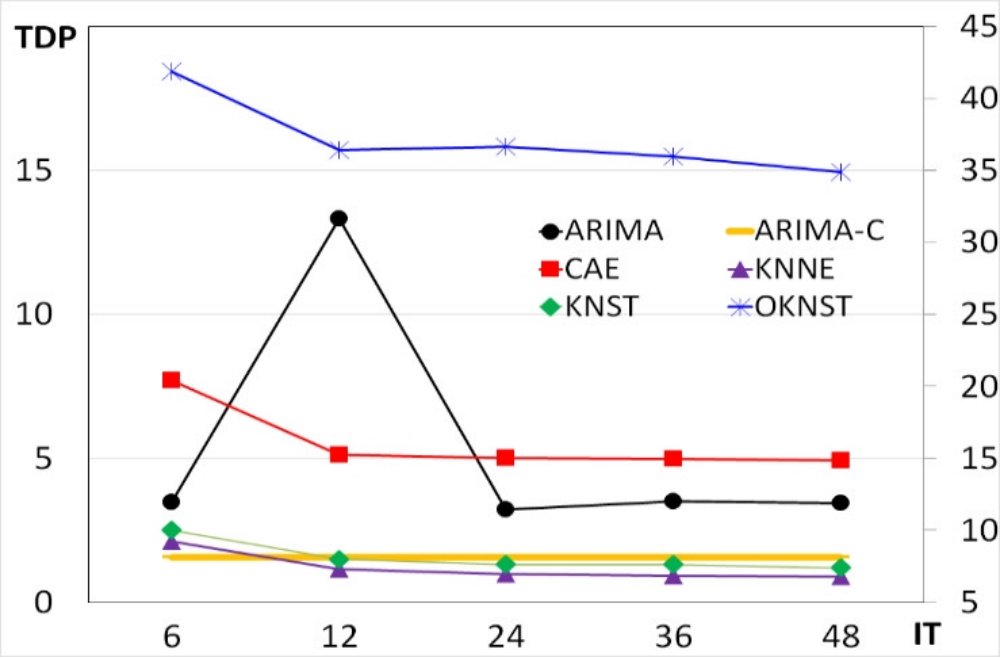


Fig. 12. *TDP(A,G_2)/TDP(RL,G_2)*% of all algorithms.
OKNST uses a different scale from the other algorithms (its scale is marked by the right side vertical axis).

85x55mm (220 x 220 DPI)

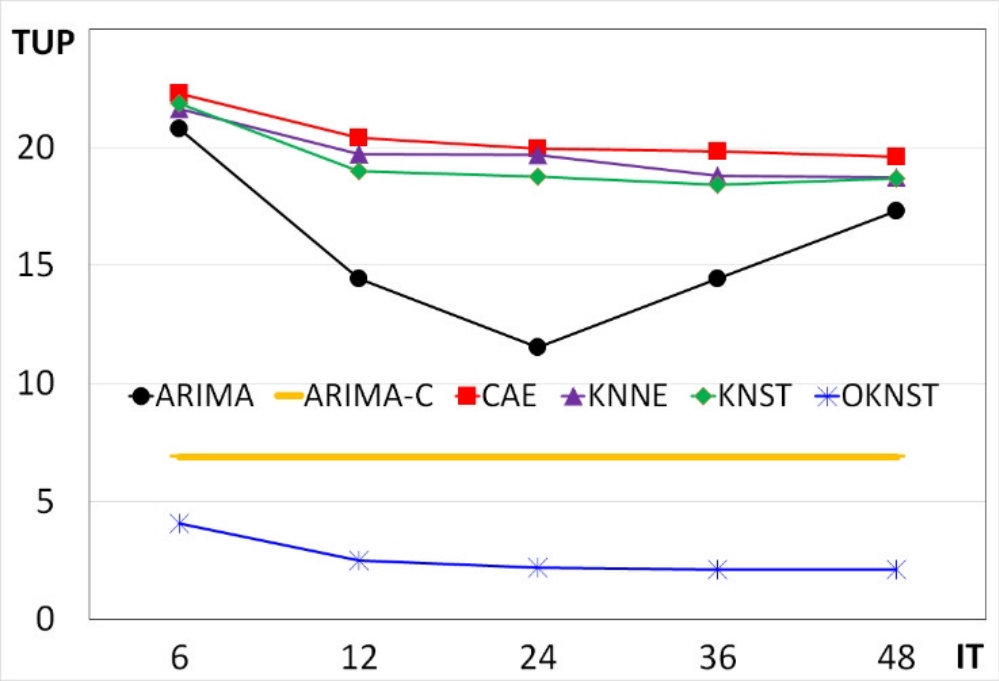
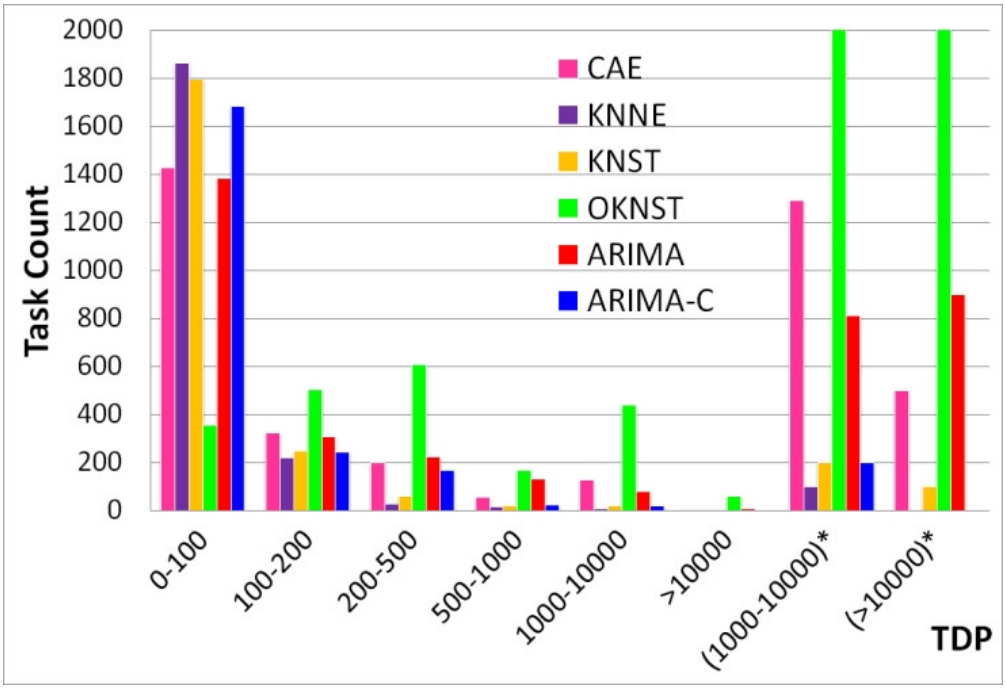


Fig. 13. *TUP(A,G_2)* of all algorithms.

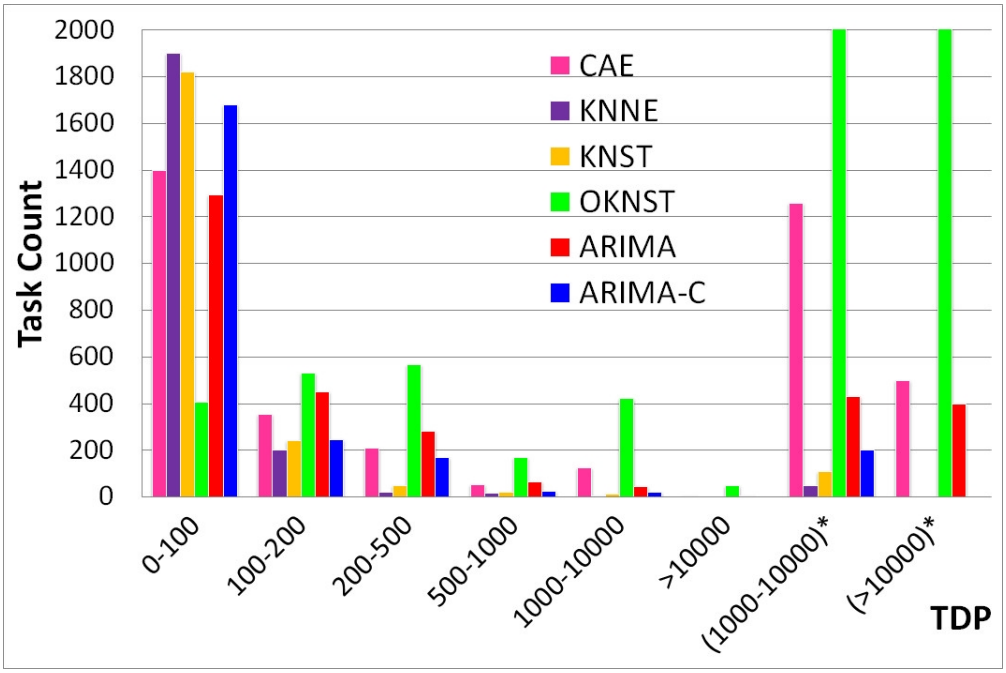
85x58mm (220 x 220 DPI)



(a) IT=12

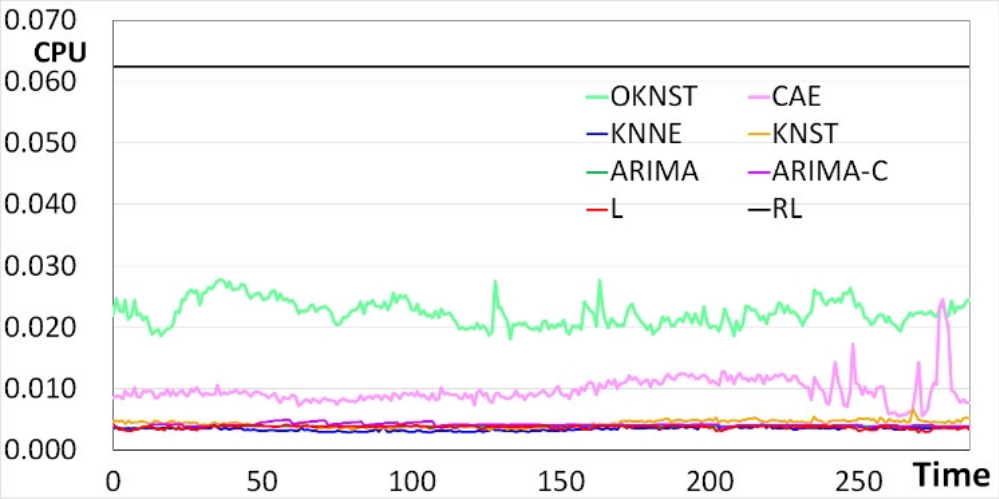
Fig. 14. Task count in each *TDP* range.
(1000-10000)*: Task Count is multiplied by 10.
(>10000)*: Task Count is multiplied by 100.

85x58mm (220 x 220 DPI)



(b) IT=48
Fig. 14. Task count in each *TDP* range.
(1000-10000)*: Task Count is multiplied by 10.
(>10000)*: Task Count is multiplied by 100.

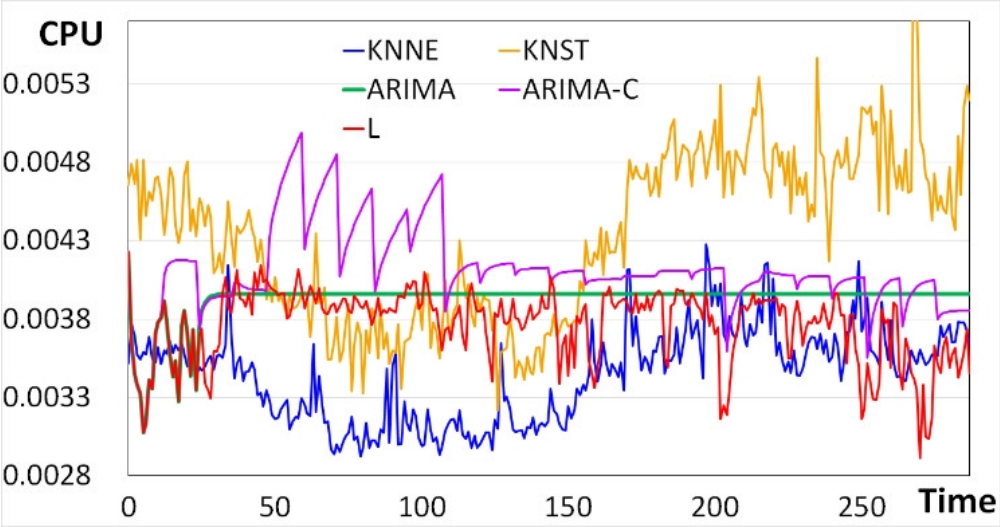
188x125mm (144 x 144 DPI)



(a) Task1

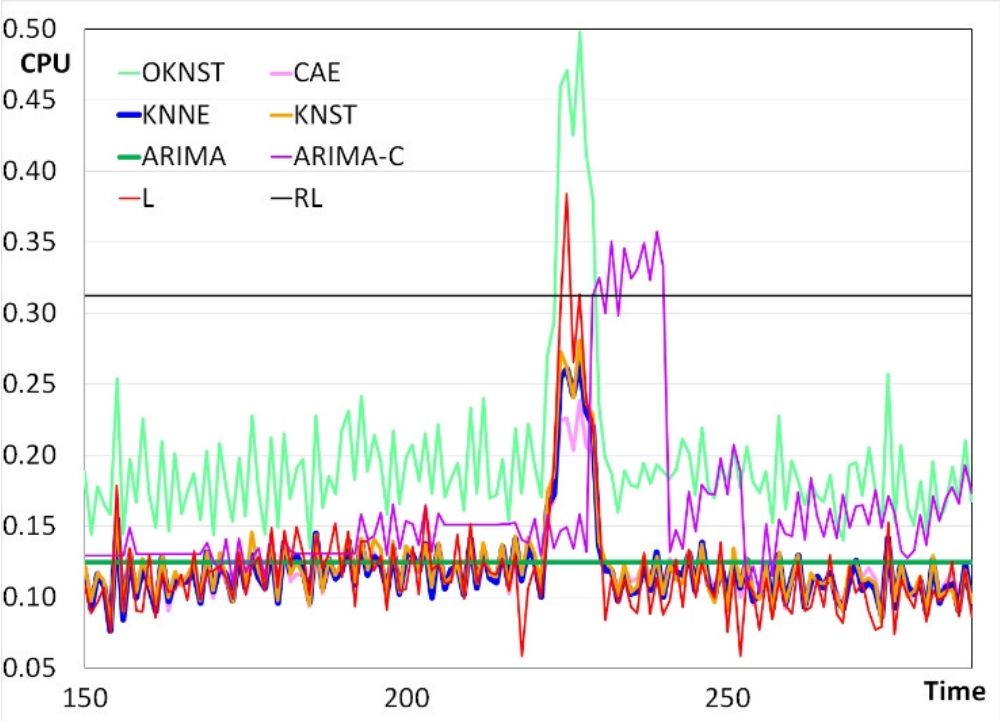
Fig. 15. Request workload, actual workload and estimation workload on single task.

86x43mm (220 x 220 DPI)



(b) Task 1 (scaled up for KNNE, KNST, ARIMA)
Fig. 15. Request workload, actual workload and estimation workload on single task.

86x45mm (220 x 220 DPI)



(c) Task 2

Fig. 15. Request workload, actual workload and estimation workload on single task.

85x61mm (220 x 220 DPI)

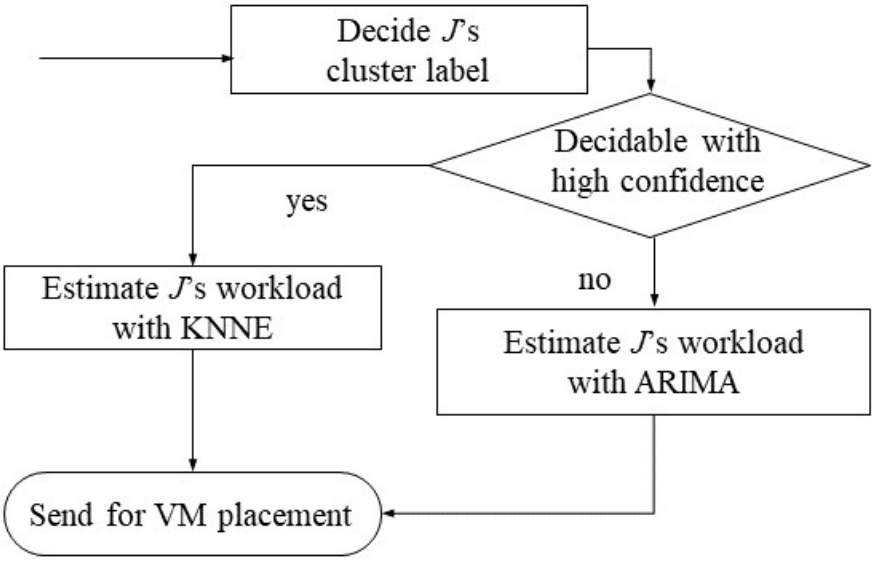


Fig. 16. Flowchart of the COMBINE algorithm at the task level.

165x118mm (96 x 96 DPI)

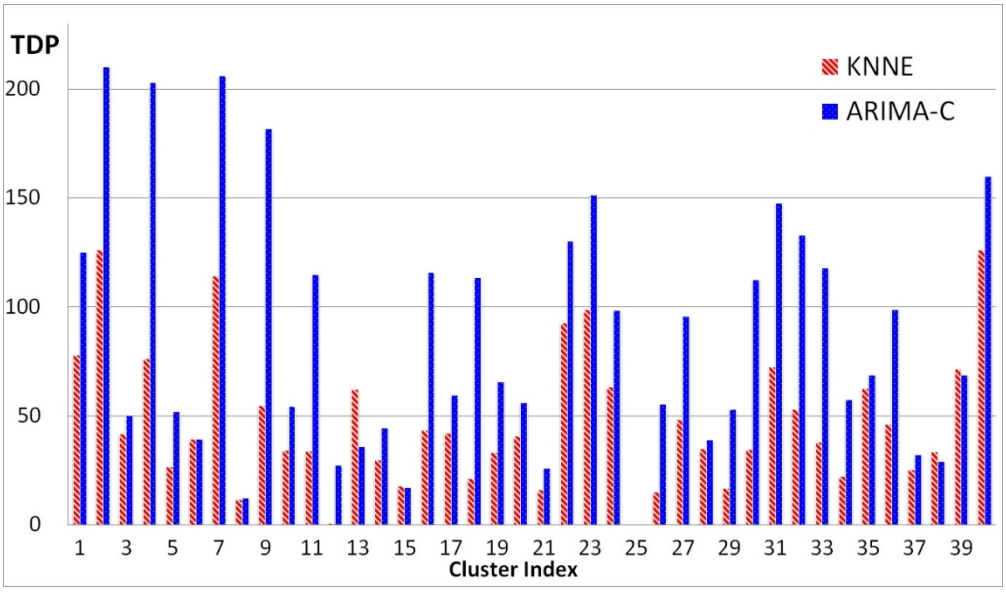


Fig. 17. *TDP* of KNNE and ARIMA-C for each clusters.

237x138mm (144 x 144 DPI)