Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH Open Access

BTP: automatic identification and prediction of tasks in data center networks

Shaojun Zou[1,2,3], Wei Ji[3] and Jiawei Huang[3*]

Abstract

Modern data centers have widely deployed lots of cluster computing applications such as MapReduce and Spark. Since the coflow/task abstraction can exactly express the requirements of cluster computing applications, various task-based solutions have been proposed to improve application-level performance. However, most of solutions require modification of the applications to obtain task information, making them impractical in many scenarios. In this paper, we propose a Bayesian decision-based Task Prediction mechanism named BTP to identify task and predict the task-size category. First, we design an automatic identification mechanism to identify tasks without manually modifying the applications. Then we leverage bayesian decision to predict the task-size category. Through a series of large-scale NS2 simulations, we demonstrate that BTP can accurately identify task and predict the task-size category. More specifically, BTP achieves 96% precision and 92% recall while obtaining accuracy by up to 98%.

Keywords: Data center, Task, Cloud computing

Introduction

To improve the communication performance of distributed data-parallel applications and user experience, a growing body of work speeds up the data transmission using task, which is defined as a collection of flows associated with the same job. These flows in tasks potentially traverse different parts of the network at different times and they need to finish before a task is considered complete [1, 2]. Unlike the flow-level transmission schemes, the task-based mechanisms are designed to reduce the completion time of all flows in tasks and achieve low task completion time.

Typically, existing task-based solutions assign the small tasks to high priority, which helps reduce the average task completion time. Although these approaches can effectively improve the network performance, they have important drawbacks. One the one hand, many schemes assume that the prior knowledge of task information such as task size can be obtained in advance. In fact, it is difficult for hosts to obtain the application-level information without modifying applications. One the other hand, the other approaches typically adopt heuristic algorithms [3] to find the local optimal solution within a limited search range. Unfortunately, we can not accurately know the arrival times of new tasks and their characteristics such as task or flow size during task scheduling. Therefore, the heuristic algorithms fail to obtain the global optimal solution, inevitably degrading the transmission performance of applications.

In the absence of task information, when one task starts to transmit data, it typically is considered as a small task and assigned high priority. Only the amount of transmitted data in one task exceeds a given threshold, the task is regarded as a large one. That is, the large tasks initially be misclassified as the small ones, making the small tasks suffer from performance penalty unnecessarily.

In this paper, we propose a Bayesian decision-based Task Prediction mechanism called BTP to automatically identify task and predict the task-size category. Here, identifying task refers to determining which flows belong to the same task according to the attributes of the flows.

*Correspondence: jiaweihuang@csu.edu.cn

[3] School of Computer Science and Engineering, Central South University, Changsha 410083, China
Full list of author information is available at the end of the article

First, we explore a machine-learning mechanism to identify tasks without manually modifying the applications. Then we leverage bayesian decision [4] to predict the task category based on the size of the task, which can provide the high accuracy of task-size prediction. The key contributions of this work are:

- We propose an application-transparent task identification mechanism that leverages machine learning to identify task without modifying applications, which facilitates practical deployment.
- We employ the bayesian decision theory to predict the task-size category. Specifically, the BTP master combines the latest and historical task information from agents to calculate the prior and conditional probability and then leverages the bayesian decision to predict the task-size category.
- By using large-scale NS2 simulations, we demonstrate that BTP can accurately identify tasks and predict the task-size category with over 98% accuracy.

The remainder of this paper is organized as follows. In "Related work" section, we summarize the related works. In "Protocol design" section, we introduce the protocol design. In "Design details" section, the detailed design of BTP is presented. In "Simulation evaluation" section, we evaluate the performance of BTP through NS2 simulations. Finally, we conclude the paper in "Conclusions" section.

Related work

In recent years, although lots of transport control protocols [5–, 6–11], load balancing schemes [12–, 13–16] and deadlock prevention mechanisms [17, 18] have been proposed to reduce flow completion time, they fail to effectively improve the application-level performance. To this end, various coflow-based scheduling mechanisms have been proposed to achieve high performance of the application level in data centers. Broadly speaking, the prior works on coflow scheduling can be classified as information-aware coflow scheduling and information-agnostic coflow scheduling.

(1) Information-aware coflow scheduling

Orchestra [19] is a centralized scheduler that mainly consists of three components: the inter-transfer controller, the cornet broadcast transfer controller, and the weighted shuffle scheduling. Specifically, the inter-transfer controller implements first-in-first-out (FIFO) and priority scheduling policies to schedule inter-coflows while both the cornet broadcast transfer controller and the weighted shuffle scheduling are designed to achieve intra-coflow scheduling. Besides, Orchestra leverages the weighted fair sharing algorithm to allocate each flow's rate according to its size.

Varys [20] abstracts the whole network as a big switch connected to all servers. Varys first proposes the smallest-effective-bottleneck-first (SEBF) heuristic to calculate the minimum coflow completion time (CCT). Then the coflow with the minimum CCT is scheduled and allocated the maximum bandwidth. Besides, Varys leverages the minimum-allocation-for-desired-duration (MADD) algorithm to address intra-coflow scheduling. For Rapier [21], it assumes all the information about coflows and topologies can be known. Then Rapier leverages the information to design optimal routing and scheduling policies.

OPTAS [22] is a decentralized solution that monitors system calls and backlogged bytes in the send-buffer to identify tiny coflows. According to the start times of tiny coflows, they are sorted in ascending order and scheduled in FIFO manner. To further reduce the coflow completion time, OPTAS assigns higher priority to tiny coflows. Similar work includes the literature [23].

(2) Information-agnostic coflow scheduling

Aalo [24] assumes the flow sizes in each coflow cannot be known prior and proposes the Coflow-Aware Least-Attained Service (CLAS) algorithm to approximate the behavior of the Least-Attained Service (LAS). Specifically, a coflow initially enters the highest priority queue. Then the coflow is removed to the lower priority queue when its sent bytes is larger than the queue's threshold. Therefore, the small coflows can obtain higher priority than large coflows. However, Aalo needs to modify the applications to obtain coflow information. On the basis of Aalo, Graviton [25] replaces the per-queue FIFO scheduling policy with the refined policies for different priority queues, which can further handle practical issues such as CoFlow starvation.

To address this issue, CODA [26] is proposed to automatically identify and schedule coflows without any application modifications. CODA leverages an incremental clustering algorithm to perform coflow identification and then designs an error-tolerant scheduling algorithm to reduce the impact of misidentifications. In addition, CODA adopts priority queues [27] and adjusts the coflow priorities according to the sent bytes of coflows.

Stream [28] is also a decentralized protocol, which leverages many-to-one and many-to-many coflow patterns to exchange information between servers. Stream emulates the conditional Shortest Job Fist (CSJF) heuristic algorithm to schedule the coflows. TaTCP [2] is a task-aware scheme that shares the flow-tardiness information through a receiver-driven coordination. Then TaTCP leverages ACK feedback to adjust the network resource and can be compatible with various TCP variants via

upgrading TCP stack on end-hosts. Similar work consists of the literatures [29–, 30–32].

Protocol design

In this section, we present the design insight and overview of BTP.

Design insight

In our design, BTP faces several key challenges that are addressed in this paper.

(1) In data center networks, the benefits of scheduling mechanisms depend on one major assumption that all data-parallel applications in a shared cluster have been modified to use the same API [26, 33]. However, it is impractical in actual situations. The major reason is that if we repeatedly modify the port information of applications in the deployed environment, it is prone to cause errors and cannot be flexibly upgraded. To address this problem, BTP uses the machine learning algorithm to identify tasks and obtain task-related information without modifying the application.

(2) For task-based scheduling mechanisms that adopt heuristic algorithms to predict task or flow size, they fail to achieve the global optimal solution. The shorter the search time of heuristic algorithms, the worse the result. To address this problem, BTP analyzes the historical data and then leverages bayesian decision to predict the size of task.

Overview of BTP

The architecture of BTP consists of following modules, as shown in Fig. 1. At a high level, BTP contains a central master and an agent on each end host. The central master is responsible for performing task identification and predicting task-size category while the agent gathers flow-level information to feed the master and backups historical task information, which is used to help the master predict task-size category.

Flow information gathering and pruning: Each host implements a BTP agent to monitor flow-level attributes such as the starting time and IP/port of each flow. Then the agent periodically sends the collected and pruned flow information to the master. This can reduce the traffic overhead between master and agent.

Task identification: According to the collected flow information, the master first mines the attributes of these flows and calculates the distance metric to obtain the distance relationship among flows. Then the master leverages the density-based clustering algorithm R-DBSCAN [34] to cluster all the flows in the data center network and obtain the corresponding clusters (i.e., the tasks). Note that the task identification is automatic and does not require application-level modification.

Prediction of task-size category: After obtaining tasks, the task-size prediction module is called to obtain the task-size category, so as to provide decision support for scheduling tasks. Specifically, the master combines the latest task information and the historical task

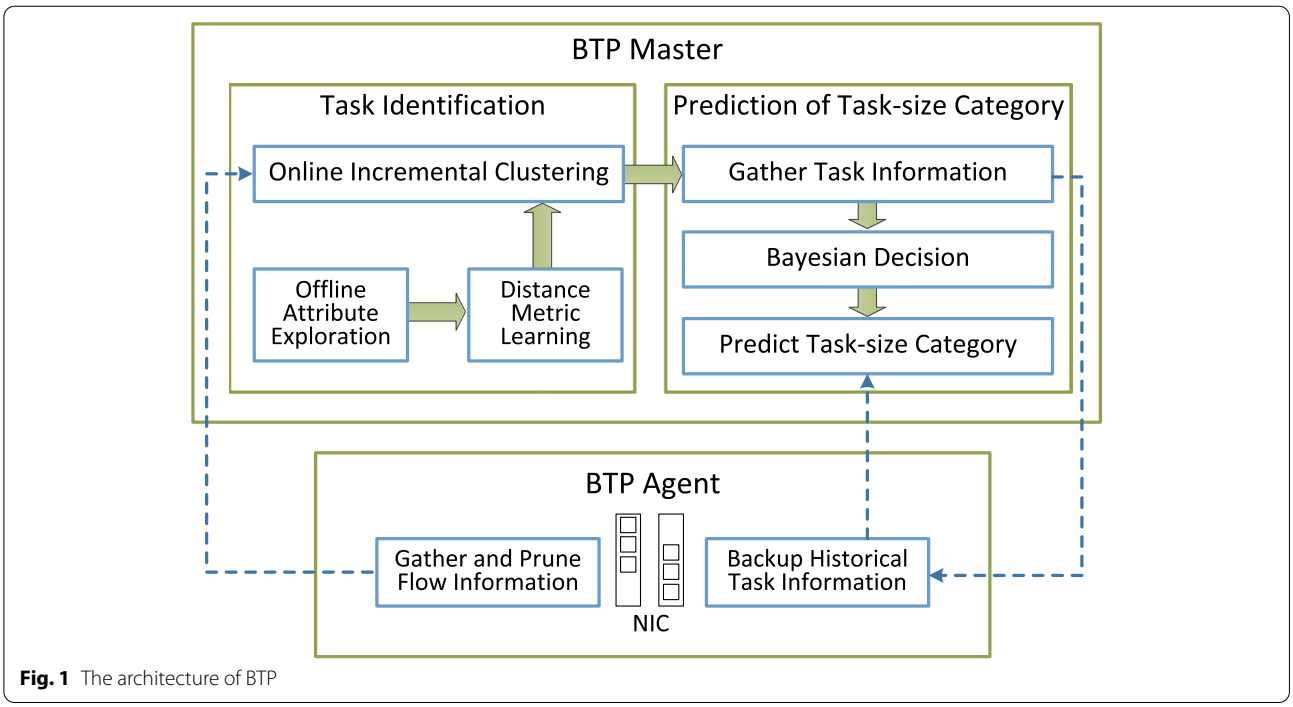


Fig. 1 The architecture of BTP

information from agents to calculate the prior probability of the task size and the corresponding conditional probability. Then, the bayesian decision is used to predict the task-size category and calculate the corresponding error rate.

Design details

In this section, we describe the detailed design of BTP, including the task identification and prediction of task-size category. Besides, the main variables used in the paper are shown in Table 1.

Task identification

In our design, task identification consists of the following three steps:

1. Attribute exploration: We first explore a set of flow, community, and application level attributes, which might be useful in task identification. For flow-level attributes, we consider flow's starting time, average packet size, variance of packet size, average arrival interval of flows. Besides, the data center can be separated into multiple service groups. Given the fact that intra-group communication is frequent while inter-group communication is rare, any two flows belonging to the same community are likely to be inside one task. Finally, we can take advantage of application designs to help task identification. For example, the port assignment rule in Spark reveals that the data transmitted to the same executor typically have the same destination IP and port.

2. Distance calculation: Given the attributes, BTP can calculate distances between flows to capture task relationships. For flows in the same task, they have smaller distances. The key problem is that we need a good metric to effectively reflect the importance of each attribute and task relationships.

In our design, we leverage distance metric learning [35] to obtain the relationship between any two flows. If the distance between two flows is smaller than the given threshold, then they belong to the same task. Otherwise, they belong to the different tasks. The distance between two flows f_i and f_j can be calculated as

$$d(f_i,f_j) = ||f_i - f_j||_A = \sqrt{(f_i - f_j)^T A (f_i - f_j)}, \quad (1)$$

where A is the distance matrix and reflects weights of different attributes.

We desire a metric where any pairs of flows in a task have small distances while any pairs of flows belonging to different tasks have distances larger than a given threshold. Here, we simplify the problem by restricting A to be diagonal and adopt Newton-Raphson method [35] solve it.

3. Identifying tasks via clustering: According to the features of aforementioned attributes and distance measurement learning, BTP master leverages the clustering algorithm R-DBSCAN [34] to identify tasks. In fact, R-DBSCAN is a variant of DBSCAN [36] and has several advantages listed below. First, although the number of tasks changes dynamically during the transmission process and cannot be determined, DBSCAN can automatically divide clusters within the ε domain without specifying the number of tasks in advance. Second, the tasks in the data center network may be composed of one single flow and R-DBSCAN can extract such tasks. Finally, R-DBSCAN uses the core objects in the cluster as an index to reduce the running time and time complexity.

Next, we introduce how the clustering algorithm works. It first scans the data set to obtain the parent object set and the child object set while establishing the relationship between the parent and child objects. Here, the parent object set refers to the set of flows that have

Table 1 Variables in BTP

Variable	Description	Variable	Description
f_i	The ith flow	$d(f_i,f_j)$	The distance between two flows f_i and f_j
A	Distance matrix T	w_{max}	Maximum threshold of task width
K_s	Threshold of small tasks	$F_s(x)$	CDF of task size
$F_w(x)$	CDF of task width	$F_l(x)$	CDF of task length
$F_{s,w}(x)$	Joint distribution function of task size and task width	Δ	Threshold of task width
$P(W)$	The Probability that the task width in the trace is within the range of $[K,\Delta]$	$P(SW)$	The probability that the task width is within the range of $[K,\Delta]$ and it is a small task
L	Task length	S	Task size
w	Task width	W	Task width set
$P(S=0)$	The small task probability	$P(S=1)$	The large task probability
$P(w_i\|S=0)$	Conditional probability of task width with small task	$P(w_i\|S=1)$	Conditional probability of task width with large task
$P(S=0\|w_i)$	Conditional probability of small task with task width w_i	$P(S=1\|w_i)$	Conditional probability of large task with task width w_i

the same traffic characteristics while the child object set is defined as the set of flows that are not classified. Then the parent object set is regarded as the data set and the master calls the DBSCAN to obtain the cluster of the parent object. Finally, based on the relationship between the parent and child objects, the master can obtain all the child objects in the identified parent object and merges them into a cluster, which is the task.

The prediction mechanism of task-size category

When the task is obtained through the task identification, we can get the task width. When the task is in transit, both its length and size are collected and put them into the data set. According to these information, we use bayesian decision to predict the task-size category.

Table 2 shows the typical traffic characteristics of tasks in the data center network. These characteristics are extracted from the Hive/MapReduce [37] traces, which are collected from the Facebook's data center with 3,000 machines and 150 ToR switches [38]. As shown in Table 2, although the proportion of short-narrow tasks is larger than other task types, the proportion of short-narrow tasks' bytes is very small (only 0.01%) and we can ignore the data of short-narrow tasks. Moreover, since the proportions of long-narrow tasks (16%), long-wide tasks(15%) and short-wide tasks(17%) are very similar, the classification problem is balanced. Finally, since there are 526 tasks in Facebook's data set, we can obtain the amount of data per group according to the measured result in the Table 1. More specifically, the numbers of short-narrow tasks, long-narrow tasks, long-wide tasks and short-wide tasks are 273, 84, 79, 90, respectively. In our design, the classification problem is binary-class. That is, both the task width and task size are 2-classes.

From these traffic characteristics, we observe that the number of bytes of narrow tasks only accounts for 0.68% of total byte amount, but the number of narrow tasks accounts for 68%. However, although the number of wide tasks only accounts for 32%, its byte amount accounts for 99.32%. Therefore, we think that there is a correlation between task width and size. That is, the smaller width of task is, the greater possibility that it is a small task. On the contrary, the larger width of the task, the greater possibility that it is a large task.

Table 2 Typical traffic characteristics of tasks in the data center network

Task type	1(SN)	2(LN)	3(SW)	4(LW)
Length	Short	Long	Short	Long
Width	Narrow	Narrow	Wide	Wide
% of tasks	52%	16%	15%	17%
% of Bytes	0.01%	0.67%	0.22%	99.10%

In order to verify our conjecture, we further analyze the above traces and try to find the maximum threshold w_{max} of task width, so that when the task width is smaller than w_{max}, it is a small task with high probability. If the task size is less than the threshold K_s of small task, then it is a small task. Let $F_s(x)$,$F_w(x)$,$F_l(x)$ denote the cumulative distribution functions of task size, task width and task length, respectively. Therefore, given the threshold K_s, we can calculate the maximum threshold w_{max} of the task width as

$$\begin{aligned} &min\ w_{max} \\ &subject\ to\ \frac{F_{s,w}(K_s, w_{max})}{F_w(w_{max})} \geq \alpha, \end{aligned} \tag{2}$$

Where $F_{s,w}(x)$ is the joint distribution function of task size and task width. That is, when the width of a task is less than w_{max}, it is the probability that the task is small task. If this probability is greater than α (i.e., 85%), then the minimum value of w_{max} is the expected threshold value.

Furthermore, we analyze the traces from Facebook's production data center. Figure 2 shows the distribution of task width. We observe that 80% of small tasks have a task width less than 15. Therefore, when the task width is less than 15, it is a narrow task.

Similar to the literature [20], K_s and w_{max} are set to 100 MB and 50, respectively. Note that the task width of 50 is also the threshold value for distinguishing the narrow tasks from the wide tasks. Figure 3 (a) and (b) show the distributions of task size and length, respectively.

Figure 3 (a) verifies our conjecture that narrow tasks are more likely to be small tasks while wide tasks are more likely to be large tasks. In Fig. 3 (b), we observe that there are about 80% of narrow tasks whose task lengths are less than 5MB. Therefore, if the width of the task is small, then its task length is most likely to be small. Since the task length is the size of the longest flow in the task, the task size will not exceed the product of task width and length. Therefore, the smaller the task width, the greater the probability that the task is a small task. The above is a qualitative analysis of the relationship between task width and task length. Next, we present the quantitative analysis of the relationship between task width and length.

Since small tasks are preferentially scheduled during the transmission process, we need to identify small tasks from these wide tasks. We guess that this kind of wide task is a small task potentially caused by the following reason: the task width just exceeds the threshold, but the task length is extremely small. As a result,

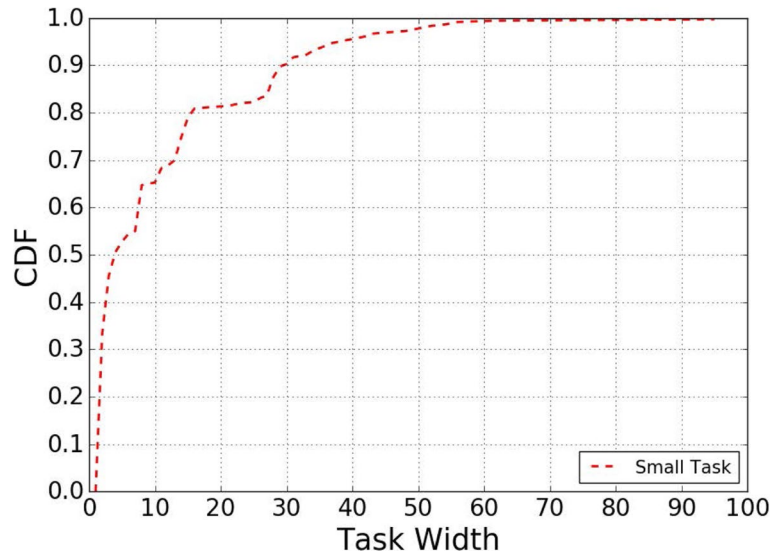


Fig. 2 CDF of task width

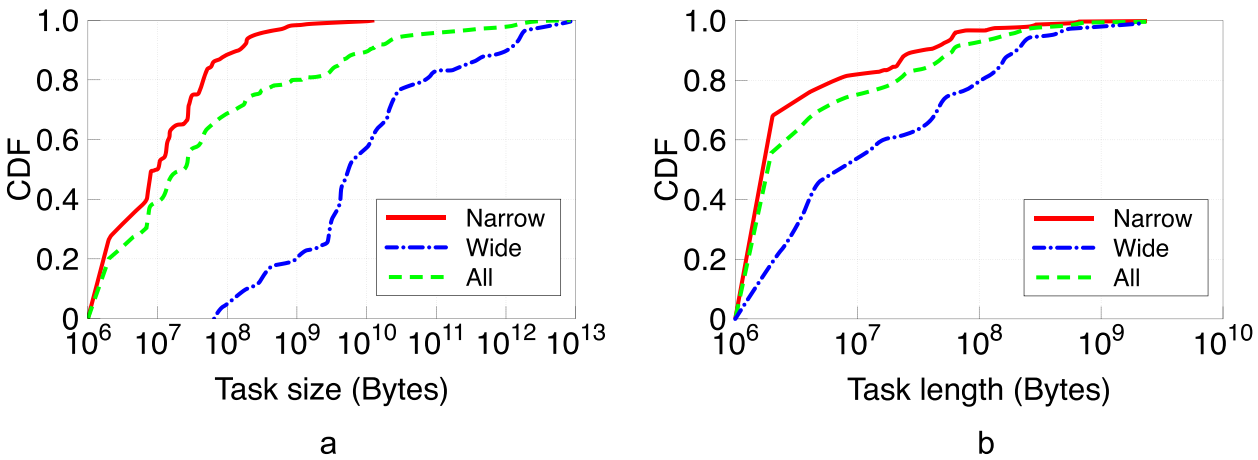


Fig. 3 Division of the task size and length with $w_{max} = 50$

the number of task bytes is small. In order to verify our guess, we further analyze the data and present the results.

Figure 4 (a) and (b) are the cumulative distribution functions of the task width of small tasks (task size < 100MB) and short tasks (task length < 5MB).

Therefore, for the wide task, we want to find the threshold Δ of task width. That is, when the task width is greater than K, it is a small task with high probability. Then we can calculate the task width threshold Δ as

$$\begin{aligned} &min\ \Delta \\ &subject\ to\ \frac{P(SW)}{P(W)} \geq \beta, \end{aligned} \tag{3}$$

where $P(W)$ represents the probability that the task width in the trace is within the range of $[K,\Delta]$ while $P(SW)$ is the probability that the task width is within the range of $[K,\Delta]$ and it is a small task. When β and the threshold of the small task are respectively set to 85% and 100MB, the threshold of task width Δ is 60. As shown in

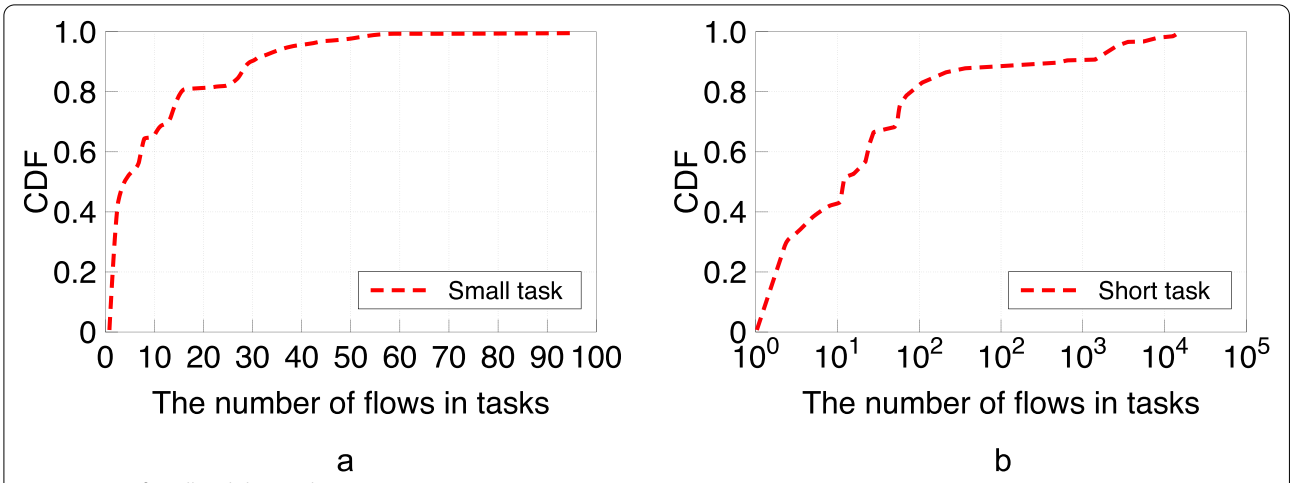


Fig. 4 CDFs of small and short tasks

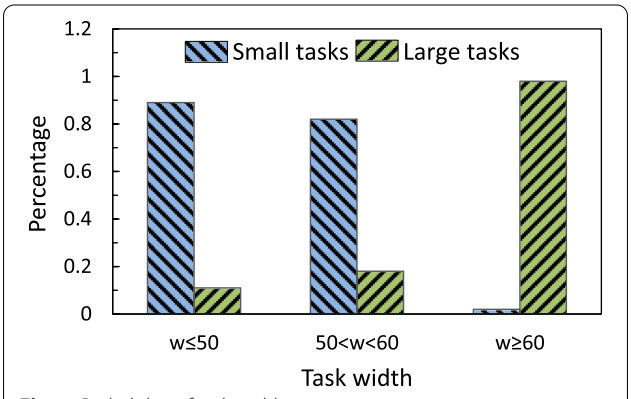


Fig. 5 Probability of task width

Fig. 5, when the width of a wide task is 50 ∼60, the probability that it is a small task is 85%.

Finally, we predict the category of task size via bayesian decision, which consists of three stages. The first stage is the preparation stage, including feature attribute selection and division. The second stage is the training stage, which trains the sample data and calculates the frequency of each category as well as the probability of each category under different conditions. The third stage is the prediction and evaluation stage that predicts the task-size category and evaluates the accuracy as well as error rate.

Feature attribute selection and division: We explore the three feature attributes: task width w, task length L and task size S. However, Fig. 3 (b) shows that there is a correlation between task length and task width. That is, the smaller the task width, the smaller the task length. Therefore, we ignore the task length attribute and only consider the task width and task size. We assume that it is a small task when S is 0 while it is a large task as the value of S is 1. As shown in Fig. 5, we divide the task width into three set intervals $W=\{w\leq 50, 50<w<60, w\geq 60\}$.

Training data: We use Facebook's real traces to test. These traces are collected from one-hour Hive/MapReduce data warehouse in Facebook's data center with 3,000 machines and 150 ToR switches. There are 526 tasks and about 7×10^5 flows in these traces (i.e., data set). In this paper, the data set is divided into training samples and test samples at the ratio of $\frac{2}{5}$ and $\frac{3}{5}$, respectively. Then we calculate the small task probability $P(S=0)$ and the large task probability $P(S=1)$ according to the sample data. Finally, we calculate the conditional probability $P(w_i|S=0)$ and $P(w_i|S=1)$ of task widths with different task-size categories, where $w_i \in W$.

The Prediction and evaluation of task-size category: Given the task width, we can calculate the probability of the task-size category (i.e., $P(S=0|w_i)$ and $P(S=1|w_i)$). In our design, if $P(S=0|w_i)$ is larger than $P(S=1|w_i)$, then it is a small task. Otherwise, it is a large task. Therefore, $P(S=0|w_i)$ and $P(S=1|w_i)$ can be expressed as

$$P(S=0|w_i)=\frac{P(w_i|S=0)\times P(S=0)}{P(w_i)}, \tag{4}$$

$$P(S=1|w_i)=\frac{P(w_i|S=1)\times P(S=1)}{P(w_i)}, \tag{5}$$

Simulation evaluation

In this section, we evaluate the performance of BTP via NS2 simulation, which is an open source network emulator with high degree of customization and has many module functions. In this test, we measure the accuracy of the automatic task identification and then estimate the accuracy of prediction method that forecasts the task-size category. The accuracy of the automatic task identification consists of two components: precision and recall. Here, we assume that there are two classes (i.e., the positive class and negative class). The positive class denotes that one flow belongs to the task while the negative class represents that the flow does not belong to the task. For the sake of convenience, let *TP*, *TN*, *FP* and *FN* denote true positive, true negative, false positive and false negative, respectively. Then the precision p is $\frac{TP}{TP+FP}$ and the recall r is equal to $\frac{TP}{TP+FN}$. Moreover, the accuracy is $\frac{TP+TN}{TP+TN+FP+FN}$.

Evaluation of automatic task identification

In this section, we use one real data set to evaluate the automatic task identification. The data set is collected in Facebook's data center with 3,000 machines and 150 racks. Moreover, the data set consists of 526 tasks (about 7×10^5 flows). Both the task size (1MB $\sim$10GB) and the number of flows within one task ($1 \sim 2\times10^4$) obey the heavy-tailed distribution. In this test, the value of the parameter ε in the task automatic identification algorithm R-DBSCAN is set to 100ms.

We first use the original data set of Facebook to verify the performance of BTP in terms of identification. The experimental result is shown in Fig. 6. We observe that BTP achieves 96% precision and 92% recall, it means that our approach can obtain good performance.

Furthermore, to evaluate BTP's broad applicability and effectiveness, we conduct our experiments using realistic workloads (i.e., Spark and Hadoop). For different workload types, the flows in intra-task have different arrival time patterns. Literature [39] points out that the arrival times of Spark traffic follow a uniform distribution and the starting times of all flows in each task is within 100ms. For Hadoop, the flows' starting times of each task are within 1000ms and follow uniformly distribution, additionally increasing 100ms exponential delay on average. In this test, we only modify the arrival times of each task and other parameters are not changed. The experimental results are shown in Fig. 7.

From Fig. 7, we observe that the precision and recall are 94% and 95% under Spark workload while Hadoop gets 92% precision and 85% recall. Besides, BTP obtains 94% precision and 90% recall under mixed workloads. Note that Spark get higher recall than Hadoop. This is because that the flows' arrival times of tasks in Hadoop workload and the clustering algorithm's ε are very similar.

Finally, we measure the error rate of prediction under different scales. Here, the error rate is defined as the ratio of the task amount that is not correctly predicted to the total number of tasks. Specifically, we can obtain the total number of tasks according to the experimental data. Compared with the predicted results, we can judge whether our prediction is correct. As a result, we get the number of tasks that are not correctly predicted and then calculate the error rate.

In this experiment, we adopt different data sets to carry out prediction. Specifically, we vary the number of tasks from 50 to 326 and the size of each task is between 1MB and 10GB. Figure 8 shows that the error rate of prediction gradually decreases as the number of tasks increases. For example, when the number of task is 326, the error rate only 9%.

Evaluation of predicting task-size category

In this section, we verify the prediction method of BTP's task-size category. We first obtain the task width and other information through the clustering algorithm of

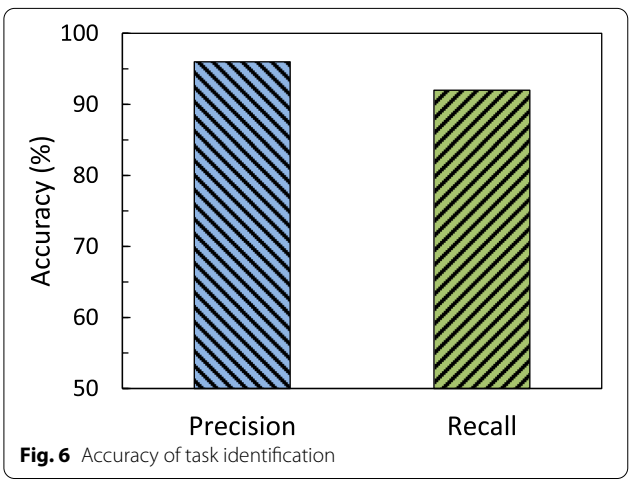


Fig. 6 Accuracy of task identification

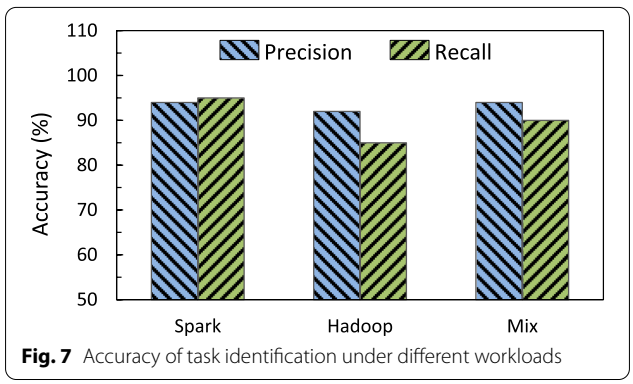


Fig. 7 Accuracy of task identification under different workloads

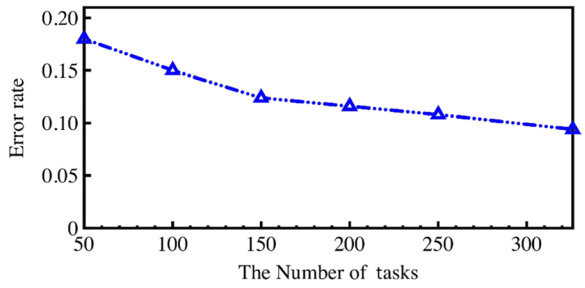


Fig. 8 Error rate under different network scales

automatic task identification. These information is used as the attribute characteristics of tasks. In addition, we calculate the prior probability and conditional probability according to the existing historical data. Finally, we leverage bayesian decision to predict the task-size category with given task width. Considering that Aalo is one typical information-agnostic coflow/task scheduling scheme that estimates the size of task according to its sent bytes, we choose Aalo to compare with BTP and test their accuracy under different task widths and workloads.

In Fig. 9, we test the accuracy of Aalo and BTP under different task widths. Here, the Narrow represents tasks with a task width less than 50, the Medium represents task's width between 50 and 60 while the Wide represents tasks with a task width greater than 60. From Fig. 9, we observe that when the task width is Narrow and Medium (i.e., the task width is less than 60), the accuracies of Aalo and BTP are very close and their values are above 80%. For Wide tasks, however, the accuracy of BTP is as high as 98.3% while the accuracy of Aalo is less than 5%. The major reason is that, for Aalo, one task initially is regarded as a small task and enters the highest priority queue. Then the task is removed to the lower priority queue when its sent bytes is larger than the threshold of current priority queue. Besides, most of Narrow and Medium tasks are small tasks. Therefore, Aalo can obtain high accuracy as the task width is Narrow and Medium. Unfortunately, since most Wide tasks are large tasks, but Aalo initially treats them as small tasks, resulting in low accuracy. On the other hand, since BTP combines the latest task information and the historical task information from agents to calculate the prior probability of the task size while leveraging the bayesian decision to predict the task size, it obtains high accuracy under all task widths.

Furthermore, we evaluate the accuracy of the prediction of task-size category under different workloads (Spark, Hadoop and their mixed traffic). Figure 10 shows that BTP achieves higher accuracy than Aalo under all workloads and its all accuracy is over 90%. This is because that Aalo is an information-agnostic mechanism and cannot obtain a high accuracy rate based on the number of bytes sent by end host. Fortunately, BTP leverages the latest and historical task information to calculate the prior probability of the task size and then adopts the bayesian decision to estimate the task size. As expected, BTP obtains high accuracy under all task workloads.

Performance under realistic workload

In this section, we conduct large-scale NS2 simulations to evaluate BTP's performance. As shown in Fig.11, we adopt a popular leaf-spine topology with 10 leaf switches and 10 spine switches. The network has 200 servers and each leaf switch is attached to 20 servers with 10Gbps links. Besides, the round-trip time and buffer size of a single interface are 100 μs and 200 packets [40], respectively. Here, we compare BTP with Varys [20] and Aalo [24]. Varys assume that coflow sizes are known while Aalo schedules coflows without any prior knowledge.

In this scenario, we use a typical cloud storage application to test BTP's performance. In our

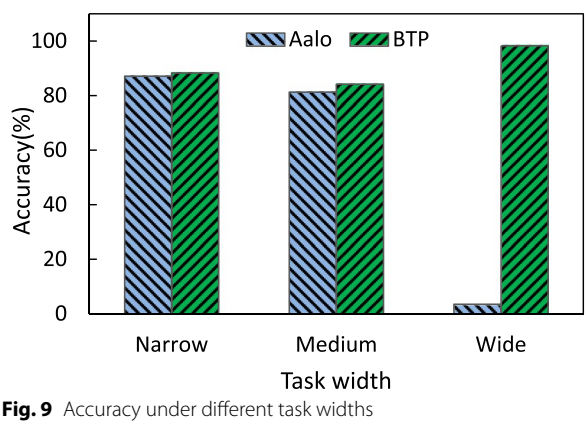


Fig. 9 Accuracy under different task widths

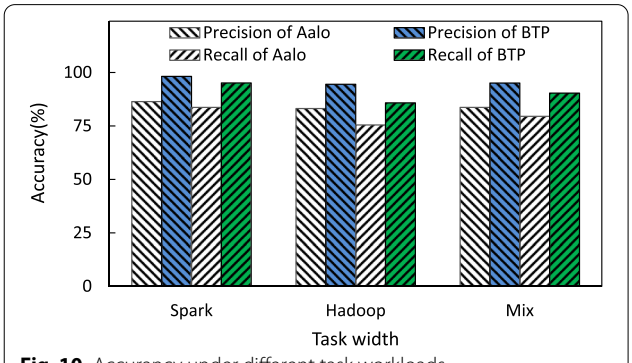


Fig. 10 Accurancy under different task workloads

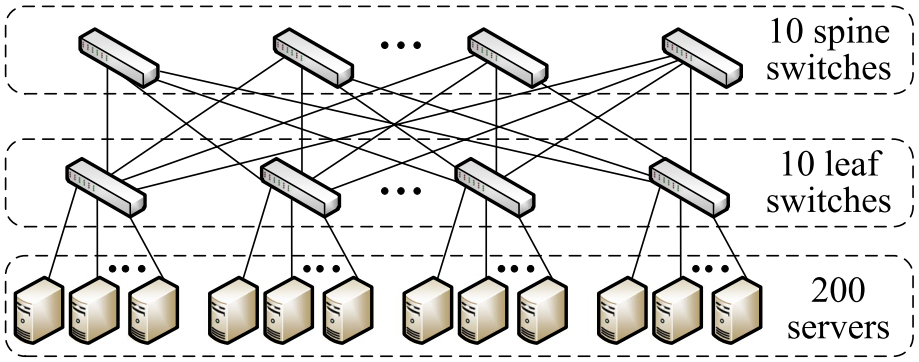
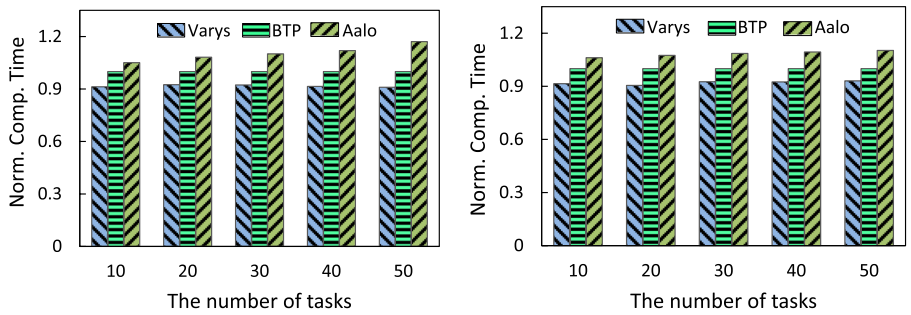


Fig. 11 Leaf-Spine topology



(a) Normalized average completion time of small tasks

(b) Normalized 99th-ile completion time of small tasks

Fig. 12 Performance under cloud storage workload

experiments, we vary the number of tasks and each task consists of 30 flows. Similar to [24, 41–44], we measure the performance metric as the completion time of a scheme normalized by BTP's completion time. It means that if the normalized completion time of a scheme is greater (smaller) than 1, it is slower (faster) than BTP.

Figure 12 shows that Varys obtains the lowest task completion time than other schemes while Aalo gets largest task completion time. This is because Varys knows the sizes of the tasks in advance and can accurately schedule small tasks, which helps reduce the task completion time. However, Aalo schedules coflows without any prior knowledge. Each task initially is regarded as a small task, which enters high priority queue and increase task completion time. Fortunately, BTP can accurately identify task and predict the task-size category. As a result, most small tasks can be scheduled first, making BTP obtain better performance the Aalo. Moreover BTP has comparable performance to Varys and does not modify application, which facilitates practical deployment.

Conclusions

This work presents the design and implementation of BTP, which automatically identifies task and predicts the task-size category. BTP employs a clustering algorithm to achieve an application-transparent task identification and then predicts the task-size category based on bayesian decision. Our experimental results show that BTP can achieve accuracy by up to 98% .

Acknowledgements
Not applicable.

Authors' contributions
All authors have participated in conception and design, or analysis and interpretation of this paper. The author(s) read and approved the final manuscript.

Funding
This work is supported by the National Natural Science Foundation of China (62102047, 62132022, 61872387, 61872403), Key Research and Development Program of Hunan (2022WK2005), Natural Science Foundation of Hunan Province, China (2021JJ30867), Project of Foreign Cultural and Educational Expert (G20190018003), the Hunan Province Key Laboratory of Industrial Internet Technology and Security (2019TP1011).

Availability of data and materials
Not applicable.

Declarations

Competing interests
The authors declare that they have no competing interests.

Author details
[1]School of Computer Science and Engineering, Changsha University, Changsha 410022, China. [2]Hunan Province Key Laboratory of Industrial Internet Technology and Security, Changsha University, Changsha 410022, China. [3]School of Computer Science and Engineering, Central South University, Changsha 410083, China.

Received: 24 June 2021 Accepted: 13 August 2022
Published online: 01 September 2022

References
1 Dogar FR, Karagiannis T, Ballani H, Rowstron A (2014) Decentralized task-aware scheduling for data center networks In: Proc. ACM SIGCOMM, New York 431–442.
2 Liu S, Huang J, Zhou Y, Wang J, He T (2019) Task-aware TCP in data center networks. IEEE/ACM Trans Networking 27(1):389–404.
3 Bai W, Chen L, Chen K, et al. (2017) PIAS: Practical information-agnostic flow scheduling for commodity data centers. IEEE/ACM Trans Networking 25(4):1954–1967.
4 Yuille AL, Bthoff HH (1996) Bayesian decision theory and psychophysics. Percept Bayesian Infer 11(4):123–161.
5 Huang J, Huang Y, Wang J, He T (2020) Adjusting packet size to mitigate TCP Incast in data center networks with COTS switches. IEEE Trans Cloud Comput 8(3):749–763.
6 Zou S, Huang J, Wang J, He T (2020) Flow-aware adaptive pacing to mitigate TCP incast in data center networks. IEEE/ACM Trans Networking 29(1):134–147.
7 Zhang T, Huang J, Chen K, Wang J, Chen J, Pan Y, Min G (2020) Rethinking fast and friendly transport in data center networks. IEEE/ACM Trans Networking 28(5):2364–2377.
8 Huang J, Li W, Li Q, Zhang T, Dong P, Wang J (2020) Tuning high flow concurrency for MPTCP in data center networks. J Cloud Comput 9(1):1–15.
9 Huang J, Li S, Han R, Wang J (2019) Receiver-driven fair congestion control for TCP outcast in data center networks. J Netw Comput Appl 131:75–88.
10 Zeng G, Bai W, Chen G, Chen K, Han D, Zhu Y, Cui L (2019) Congestion control for cross-datacenter networks In: Proc. IEEE ICNP, Piscataway 1–12.
11 Hu S, Bai W, Zeng G, Wang Z, Qiao B, Chen K, Tan K, Wang Y (2020) Aeolus: A building block for proactive transport in datacenters In: Proc. ACM SIGCOMM, New York 1–13.
12 Zou S, Huang J, Jiang W, Wang J (2020) Achieving high utilization of flowletbased load balancing in data center networks. Futur Gener Comput Syst 108(2020):546–559.
13 Liu J, Huang J, Lv W, Wang J (2020) APS: Adaptive packet spraying to isolate mix-flows in data center network. IEEE Trans Cloud Comput:1–14. https://doi.org/10.1109/TCC.2020.2985037.
14 Hu J, Huang J, Lv W, Zhou Y, Wang J, He T (2019) CAPS: Coding-based adaptive packet spraying to reduce flow completion time in data center. IEEE/ACM Trans Networking 27(6):2338–2353.
15 Huang J, Lv W, Li W, Wang J, He T (2021) Mitigating packet reordering for random packet spraying in data center networks. IEEE/ACM Trans Networking 29(3):1183–1196.
16 Zou S, Huang J, Wang J, He T (2021) RMC: Reordering marking and coding for fine-grained load balancing in data centers. IEEE Trans Commun 69(12):8363–8374.
17 Hu S, Zhu Y, Cheng P, Guo C, Kun Tan J, Padhye K (2016) Chen, Deadlocks in datacenter networks: why do they form, and how to avoid them In: Proc. ACM HotNets, New York 1–7.
18 Hu S, Zhu Y, Cheng P, Guo C, Tan K, Padhye J, Chen K (2017) Tagger: Practical PFC deadlock prevention in data center networks In: Proc. ACM CoNEXT, New York 451–463.
19 Chowdhury M, Zaharia M, Ma J, Jordan MI, Stoica I (2011) Managing data transfers in computer clusters with Orchestra In: Proc. ACM SIGCOMM, New York 98–109.
20 Chowdhury M, Zhong Y, Stoica I (2014) Efficient coflow scheduling with Varys In: Proc. ACM SIGCOMM, New York 443–454.
21 Zhao Y, Chen K, Bai W, et al. (2015) RAPIER: Integrating routing and scheduling for coflow-aware data center networks In: Proc. IEEE INFOCOM, Piscataway 424–432.
22 Li Z, Zhang Y, Li D, Chen K, Peng Y (2016) OPTAS: Decentralized flow monitoring and scheduling for tiny tasks In: Proc. IEEE INFOCOM, Piscataway 1–9.
23 Li Z, Shen H (2022) Co-Scheduler: A coflow-aware data-parallel job scheduler in hybrid electrical/optical datacenter networks. IEEE/ACM Trans Networking:1–14. https://doi.org/10.1109/TNET.2022.3143232.
24 Chowdhury M, Stoica I (2015) Efficient coflow scheduling without prior knowledge In: Proc. ACM SIGCOMM, 393–406.
25 Jajoo A, Gandhi R, Hu YC (2016) Graviton: Twisting space and time to speed-up coflows In: Proc. USENIX HotCloud, 1–6.
26 Zhang H, Chen L, Yi B, et al. (2016) CODA: Toward automatically identifying and scheduling coflows in the dark In: Proc. ACM SIGCOMM, 160–173.
27 Chen L, Lingys J, Chen K, Liu F (2018) AuTO: Scaling deep reinforcement learning to enable datacenter-scale automatic traffic optimization In: Proc. ACM SIGCOMM, 191–205.
28 Susanto H, Jin H, Chen K (2016) Stream: Decentralized opportunistic inter-coflow scheduling for datacenter networks In: Proc. IEEE ICNP, 1–10.
29 Wei Z, Guo S, Liu G, Yang Y (2021) Coflow scheduling with unknown prior information in data center networks In: Proc. IEEE ICC, 1–6.
30 Liu L, Gao C, Wang P, Huang H, Li J, Xu H, Zhang W (2021) Bottleneck-aware non-clairvoyant coflow scheduling with Fai. IEEE Trans Cloud Comput:1–14. https://doi.org/10.1109/TCC.2021.3128360.
31 Zhang F, Tang Y, Shan D, Wang H, Hu C (2021) RICH: Strategy-proof and efficient coflow scheduling in non-cooperative environments. J Netw Comput 196:1–13.
32 Sun P, Guo Z, Wang J, Li J, Lan J, Hu Y (2021) Deepweave: Accelerating job completion time with deep reinforcement learning-based coflow scheduling In: Proc. International Joint Conferences on Artificial Intelligence, New York 3314–3320.
33 Yi B, Chen L, Xia J, Chen K (2017) Towards zero copy dataflows using rdma In: Proc. ACM SIGCOMM Poster and Demo, New York 28–30.
34 Viswanath P, Babu VS (2009) Rough-DBSCAN: A fast hybrid density based clustering method for large data sets. Pattern Recogn Lett, Elsevier 30(16):1477–1488.
35 Xing EP, Ng AY, Jordan MI, Russell SJ (2002) Distance metric learning with application to clustering with side-information In: Proc. ACM NIPS, 521–528.
36 Ester M, Kriegel H, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise In: Proc. ACM KDD, New York 226–231.
37 Chen L, Chen K, Zhu J, Yu M, Porter G, Qiao C, Zhong S (2017) Enabling wide-spread communications on optical fabric with MegaSwitch In: Proc. USENIX NSDI, New York 577–593.
38 Coflow benchmark based on facebook traces. https://github.com/coflow/coflow-benchmark. Accessed May 2017.
39 Meisner D, Sadler CM, Barroso LA, Weber W, Wenisch TF (2011) Power management of online data-intensive services In: Proc. International Symposium on Computer Architecture (ISCA), New York 319–330.
40 Bai W, Chen L, Chen K, Wu H (2016) Enabling ECN in multi-service multi-queue data centers In: Proc. USENIX NSDI, New York 537–549.
41 Zhang H, Zhang J, Bai W, Chen K, Chowdhury M (2017) Resilient datacenter load balancing in the wild In: Proc. ACM SIGCOMM, New York 253–266.
42 Zhang J, Bai W, Chen K (2019) Enabling ECN for datacenter networks with RTT variations In: ACM CoNEXT, New York 233–245.
43 Bai W, Chen K, Chen L, Kim C, Wu H (2016) Enabling ECN over generic packet scheduling In: Proc. ACM CoNEXT, New York 191–204.
44 Bai W, Hu S, Chen K, Tan K, Xiong Y (2020) One more config is enough: saving (DC)TCP for high-speed extremely shallow-buffered datacenters. IEEE/ACM Trans Networking 29(2):489–502.

Publisher's Note