2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing

Pota: Maximizing Profit for Task-level Scheduling for Data Center Networks

Xiaoyi Tao, Heng Qi, Wenxin Li, Keqiu Li
School of Computer Science and Technology
Dalian University of Technology
DaLian, China
keqiu@dlut.edu.cn

Yingwei Jin
School of Management
Dalian University of Technology
DaLian, China
jinyw67@dlut.edu.cn

***Abstract*—Scheduling is one key issue in Data Center Networks (DCN). Earlier research work usually focuses on flow-level scheduling, while more and more people are aware of the benefits of task-level scheduling in recent years. Most existing task-level scheduling methods schedule flows of one task together in order to reduce average completion time. However, few works discuss the efficient task-level scheduling in the view of the profit of tasks.**

To address this problem, we propose a novel task-level scheduling method named Pota, whose target is to maximize the profit of completing tasks within their deadline. To achieve this goal, we propose a maximizing profit optimization model, and then present an efficient scheduling $\frac{1}{2}$-approximate algorithm. Based on the proposed algorithm, we design and implement Pota. We also conduct comprehensive experiments to evaluate the performance of Pota. The experimental results show that Pota can save 15% average task completion time while increasing 20% task profit.

***Keywords*-Data center networks, Task-level scheduling, Maximize Profit**

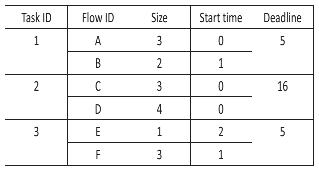
I. Introduction

Data centers are being used as the critical computing and storage infrastructures for commercial services, including web search, social network, retail system and cloud computing. These applications are under a strict limitation on latency requirements, and even a sub-second can seriously impact user experience and application profits [1]. Online applications generate a large number of requests and aggregate the results from responses computing in the back-end, since the result must wait for all of the response flows to finish or reduce user experience. This implies that a task is successfully accomplished if and only if all of flows belong to one task complete before its deadline. Accordingly, task-level scheduling methods for data center networks is significant for user experience and increasing application profits.

To the best of our knowledge, existing scheduling methods in data center networks can be classified into two categories: flow-level scheduling and task-level scheduling. On the one hand, flow-level scheduling methods focus on minimizing completion time and finishing flows within deadline only based on flow information. Fair sharing approaches approximately divide bandwidth equally on bottleneck in a fair share manner. The scheduling methods [2] [3] are based on transport level rate control, which reduce the number of flows missing deadlines with weakness of first come first serve (FCFS). Centralized flow priority scheduling methods [4] [5] reduce flow completion time. With the aim of improving network resource efficiency, Hedera [6] dynamically schedules elephant flows and short flows. The integrity of task is ignored by flow-level scheduling. Each task in data centers contains hundreds of flows, all of which need to be finished before a task is considered to be completed. Only flow-level scheduling could be in a situation that most flows in one task completed waiting for lag flows completion or returning results to users with performance damage. On the another hand, task-level scheduling methods organize flows of one task and schedule them together in order to reduce average completion time. Baraat [7] works on decentralized task-level schedule and schedules flows in one task together. Coflow [8] abstracts the network plane and Varys [9] schedules coflows to reduce coflow completion time (CCT).

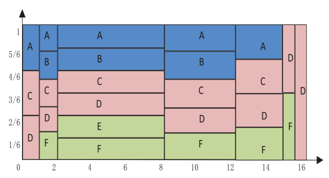
Prior works focus on reducing flow completion time, from the view of user experiment and application profit, task overall performance are the important criterions for data centers. However, both flow-level and task-level scheduling methods ignore the overall task performance. We define profit as a variable to describe task performance. Meanwhile, profit is a representative of task completion condition and network efficiency, thus take advantage of task profit in task-level scheduling is feasible. Flows belonging to different applications have different characteristic on flow sizes and latency. These have motivated task-level problems of scheduling.

In this paper, we propose a task-level scheduling method to improve the task performance, which is measured with task profit. To ease presentation, we present an example to demonstrate potential benefits of task-level scheduling methods by applying existing flow-level methods. Task-level scheduling methods are based on both task and flow priorities. As shown in Figure 1, we consider that there are three tasks with six flows. Each flow is represented as a 5-tuple [task ID, flow ID, size, starttime, deadline]. Suppose that these flows are transferred in one bottleneck

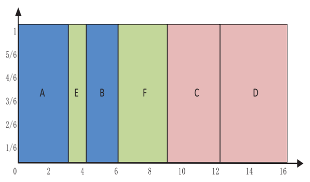


Task ID	Flow ID	Size	Start time	Deadline
1	A	3	0	5
	B	2	1	
2	C	3	0	16
	D	4	0	
3	E	1	2	5
	F	3	1	

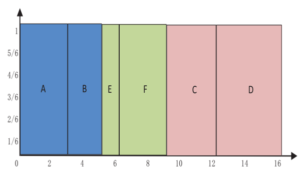
(a) The concurrent flows conditions for one bottleneck link



(b) Bandwidth fair share



(c) Flow-level priority scheduling



(d) Task-level priority scheduling

Figure 1. An example of current works vs task-level schedule

link, and the network resource cannot satisfy all the flows deadline demands. In traditional solutions, fair sharing is the main method which is shown in Figure 1(b). Here, the flow completion time of the six flows are (44/3, 12, 44/3, 16, 8, 46/3) respectively. Comparing the finish time with deadline 1(a), only flows C and D meet their deadlines, and the average flow completion time is 13.4 by applying the fair sharing. In Figure 1(c), flows are scheduled by priorities that are associated with deadlines. As shown in the Figure 1(a), both task 1 and 3 miss their deadlines. Deadline priority reduces the average completion time to 7.7. Comparing to fair sharing method, it saves 42% on flow completion time. Flow-level priority scheduling methods schedule flows according to the flow priority without task information. In this case, both task 1 and 3 complete only one flow. This means that they obtain zero profit. In task-level priority scheduling as shown in Figure 1(d), the average flow completion time is 7.8 which is proximal to priority scheduling methods. However, the average task completion time of flow-level priority scheduling is 10 while task-level scheduling is 9.6. Moreover, task-level method reduces the number of switches among the flows which might bring some lag during switching in this scenario. Task 1 meets its deadline in task-level scheduling, therefore task-level scheduling obtains more profit than flow-level scheduling.

According to above analysis, we design Pota (maximizing profit for task-level), a method that targets at maximizing profit of completing tasks within their deadline, which focuses on task-level scheduling in data center networks. Pota utilizes centralized controller, which has knowledge of task-level information on task number, task requirement and task deadline. The core of Pota is an $\frac{1}{2}$-approximation algorithm on maximizing profit according to task time and task profit, which helps controller make decisions on schedule sequences. In Pota, the controller identifies the flows that belong to the same tasks and dynamically modifies their priories with the aim of guaranteeing task integrity. Pota schedules tasks in a priority manner according to the policy computed by the controller. Through extensive simulation, we finds that Pota provides benefits over existing benefits on task completion rate, task profit and task completion time.

To sum up, our main contributions of this paper are as follows:

- We introduce a maximizing profit optimization model to design an efficient task-level scheduling method based on deadline, priority and task requirements. Specifically our model is capable of improving flow-level and application profits.
- We design and implement Pota, a task-level flow scheduling method. Guided by the profit optimization model, the key idea of Pota is an approximate method that computes the scheduling policy based on the priority, task completion condition and task requirement.
- We conduct comprehensive experiment to evaluate the performance of Pota, compared with existing priority flow scheduling solutions under typical data center traffic [10]. Pota outperforms the flow-level priority scheduling in data center networks. Pota can save 15% average task completion time. Meanwhile, it increases 20% task profit.

The rest of this paper is organized as follows. In Section II, we briefly introduce the work model of Pota. In Section III, we show the detail design of Pota. We present the evaluation in Section IV. In Section V, we discuss the related work of this paper. Section VI concludes the paper and our future work.

II. System Model

In this section, we first present an overview of maximizing profit problem. Then, we describe the detail of our model.

A. Problem Statement

Many data center applications generate complex and a large number of tasks such as MapReduce tasks. From the nature of tasks, these tasks consist of many flows. Meanwhile, each task has a strict limitation on deadline and all of its flows need to be finished before the deadline. Tasks belonging to different types of applications have diverse requirements on deadline. Task performance is an important factor in quality of service. In order to evaluate task performance, we define profit as a parameter to measure the task performance. Considering task performance, we make efforts to solve a task-level scheduling problem that subjects to task deadline and task profit. Generally a problem on task-level scheduling is difficult to solve. To schedule tasks efficiently, we resort to the centralized controller.

B. Model Formulation

We formulate a task-level model in data center and define variety kinds of tasks such as background flows, search engine and map reduce. We assume that task size, deadline and priority information is known. Each task in our model has an identifier denoted by the symbol t_i. We use $f_{i,n}$ to denote flows in task t_i, n_i is the number of flows in t_i. The arriving time of task t_i is a_i, and d_i indicates the deadline of t_i. The duration of task t_i is the interval between a_i and d_i. Since flow $f_{i,j}$ belongs to task t_i, they have the same deadline of d_i. Flows in one task have different arriving time. Each flow in tasks has a value on accomplishment. If a flow completes before its deadline, then the data center can obtain the flow profit. In our model, we denote $\alpha_{i,j}(t)$ as a binary variable that indicates whether to schedule flow $f_{i,j}$ at current time slot. Here, we suppose that once a flow is scheduled before its deadline, we consider that it has been completed. While network resource is limited, we denote C as the capacity of network that can be allocated to schedule flows. Let $s_{i,j}$ indicate the size of flow $f_{i,j}$. Each task has a weight ω_i associated with task value. Symbol ρ_i denotes the priority of task t_i, for example realtime tasks have higher priority than background tasks. Applications in data centers pay different prices for their resource. The profit of tasks belonging to these applications is affected by these prices which can be denoted by task weight ω, and ρ is an indictor of the nature of applications. Therefore, the profit of task t_i P_i as shown in Equation (1) is determined by task priority ρ_i and task weight ω_i.

$$P_i = \omega_i \rho_i. \tag{1}$$

Flow profit $P_{i,j}$ is determined by two parts: task profit P_i and task completion condition. Here, we let the task completion condition be denoted by γ_i, which satisfies $\gamma_i > 1$ associate with time urgency of task t_i. And γ_i can change over time slots. $P_{i,j}$ can be dynamically modified according to current time and deadline which means schedule some urgent tasks' flows can bring in higher profit.

$$P_{i,j} = \gamma_i * P_i. \tag{2}$$

The profit is gained by successfully schedule a flow as

$$G_{i,j} = P_{i,j}\alpha_{i,j}(t). \tag{3}$$

We are aiming to introduce an approximate schedule policy to maximizing overall task profit with deadline limitation for data center networks. Task profit and scheduling revenue is formulated in equation 2 and 3. Here we use the summation of flow profit in one task to denote the overall task profit.

The program is presented in Equations (4) - (7). The objective function in Equation (4) defines the profit getting from schedule tasks, in the meanwhile partially scheduling tasks cannot get partial profit. Equation (5) is a limitation on task integrity which ensures the task completion in legal time, and Equation (6) is the network capacity limitation. In Equation (5), all of the finished flows during legal time subject to the flow number n_i of task i. Network capacity C is an abstraction of network resource that can be allocated to flows for transporting. All of the scheduled flows in the network strict subject to network capacity C. We consider the profit of data center to be maximum, in order to picture this situation we introduce a strategy that the priorities of flows can change with tight deadline and priorities reversely effect the profit of the flows. Since in one task the last several flow seriously impact task overall performance.

$$Max \quad \sum_{i=1}^{n}\sum_{j=1}^{n_i} G_{i,j}, \tag{4}$$

$$s.t. \quad \sum_{a_i}^{d_i}\sum_{j=1}^{n_i}\alpha_{i,j}(t) = n_i, \tag{5}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n_i} s_{i,j}\alpha_{i,j}(t) \le C, \tag{6}$$

$$\alpha_{i,j}(t) \in 0,1. \qquad \forall i,j \tag{7}$$

Obviously, our model is a NP-hard problem [11]. The optimization program above cannot get the optimal solution in constant time. The optimization function is a target we make effort to solve the deadline issue and bring in more profit. We propose an approximation algorithm to obtain approximate result for scheduling policy. Our model is the first attempt to define dynamic priority during time slot in data center networks. We introduce an approximation algorithm to solve the scheduling policy in next section.

III. Profit task-level Flow Schedule (Pota)

In this section, we present the details for Pota. In Section III-A, we briefly introduce the main design of Pota. We divided Pota into two parts: controller design and switch design. In Section III-B, we introduce main function of controller. In this section, we present and analyse an appropriate algorithm to maximize overall task profit in our model. We introduce the design of switches for Pota in Section III-C.

A. Pota overview

The key insight of Pota is that the controller is aware of tasks information and generate an appropriate scheduling policy. The controller scheduling tasks upon profits. Fig.2 shows the overview of Pota. Pota employs a centralized controller to implement information collection and generate policy. From the perspective of tasks, tasks have the profits corresponding to applications. Meanwhile, flows belonging to one task have flow-level profit which can adjustment during time slots. The controller is aware of all information about networks and has the authority of networks controlling. Given this consideration, we generate a schedule policy based on overall profit and take advantage of controller function to schedule flows. The controller makes use of network global information and treats flows belonging different tasks respectively. The Pota controller has three 3 main functions: task identifier, profit computation and schedule generator. The switches in Pota execute the policy generated by the controller. In the following we mainly describe the Pota's controller designs and the approximate algorithms to maximum profit.

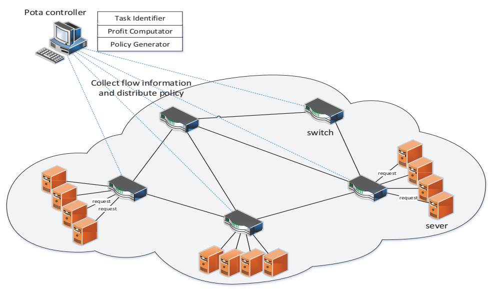


Figure 2. The framework of Pota

B. Controller design

Many data center networks are designed to have a controller, such that the network can be centralized controlled by the controller. Controller has full view of network situation and aware of flow status. In addition, data center[12] has its own administrate system which realizes the application allocation and varieties in data center. In Pota, the main function of profit compute, scheduling policy generating and priority modification are realized in the controller.

1) Task identifier: Pota controller has a counter on tasks. While tasks arrive at controller, we increase task counter and mark it as task-id. In order to identify tasks and tasks type, we design the task counter with tasks id and tasks priorities. The module of task identifier accounts for identifying both tasks and flows. In Pota, we schedule tasks based on their profits, therefore the profit of tasks is important. As shown in Equation (1), tasks profit is determined by tasks value and tasks priorities. We define task value associate with applications requirement. Therefore, the task information is crucial to Pota. The classifications of applications are basically userface latency sensitive, user-face throughput sensitive, system control and background services. System control tasks have the requirement of low latency and accuracy. While the other background flows are long-lived flows that occupy the network resource from online service. Consequently, we generally divided task priority into four levels. In application level, tasks have priorities and corresponding profit. We encodes the task priority into the task id. In this case, tasks have task id and it is corresponding tasks priorities, which provides convenience for profit calculator.

2) Profit calculator: Pota schedules tasks according to their profits and network capacity. In pota, there are two layer profits which are task profit and flow profit. Task profit is constant while flow profit can change over time. Since task have deadline, for example, on the attribute of deadline flow A has a longer deadline than flow B, and they belong to the same task-level. Only task-level profit identifer cannot distinguish flow A and flow B. A flow-level profit with deadline urgency is necessary in this situation. At the beginning, flows bring a task-level profit information and deadline information, and flow-level profit changes with the variable $gamma$. This is the initialization progress of profits. Flows belong to the same application with different task-id, they have the same task profit. Task performance for data center is based on overall task profit which is the sum of all the flows profits in one task. And the performance of network can be calculated by all tasks profits in the network.

The controller has information on task id, and tasks priorities are can be decoded from task id. The controller has knowledge of task information, and the profit of task P_i is determined by task value $omega$ and task priorities rho which is related to task deadline. As shown in Equation (1), (2) and (3), overall profit is related to task value, task priority and task completion condition. Since task-id have priority and id information, we only need to task profit and deadline information. We modify the field of vlan id to be the profit of tasks. And the priority field is used to schedule flows.

3) Policy generator: Here we introduce the main function of controller that scheduling policy generator. Approximate algorithm 1 realizes maximize profit formulation, as we define that in section II we compute a schedule policy to modify flow priorities in controller. Algorithm 1 lists all the steps in computing profit-effective, overall task profit and schedule policy. A key of our model is task profit which is related to the flow dynamic priority and task priority level. And the dynamic priorities of flows is a variable calculated by priority and time slot determined by the controller. Moreover, sdn network is a centralized network that we can take advantage of controller view to figure out the appropriate schedule policy. Next, we introduce our approximate profit algorithm.

Algorithm 1 lists all the steps followed in computing maximum profit and scheduling set $\mathcal{A}$. At first, we sort the profit-effective of the existing tasks in a descend order. In limited network resource, we schedule more profit-effective tasks which can bring in more overall profit. In step 3, we attempt to get a maximum i denoted by k. From steps 9-13, we schedule all of flows in the most profit-effective tasks which satisfy the size of appropriate tasks is smaller than current network capacity C. The network capacity C changes with the size of scheduled tasks, while C is not enough for a entire task that we schedule partial flows in the most profit-effective task at present as shown in steps 15-19.

In next loop, we schedule the left flows at first which shows in steps 6-8. The controller execute this algorithm until the capacity C reduce to zero.

Algorithm 1 Maximum Overall Profit Computation

```
Input:
    Flow $f_{i,j}$ assemble F; Profit of flow $P_{i,j}$;
    Number of flows in task i $n_i$; Size of schedule task $s_i$
Output:
    Schedule policy set $\mathcal{A}$;
    Schedule tasks profit p
1:  Sort $P_i/n_i$ in a descend sort order
2:  Scheduled tasks profit p
3:  for $i = 1; i \le n; i++$ do
4:      $P_{i,j} \leftarrow \gamma * P_i$
5:      if $C > 0 \& t \in [a_i, d_i]$ then
6:          if $n_{i-1} > 1$ then
7:              for each j from 1 to $n_{i-1}$ do
8:                  $\alpha_{i-1,j} \leftarrow 1$; $C \leftarrow C - s_{i,j}$
9:          if $s_i < C$ then
10:             for each j from 1 to $n_i$ do
11:                 $\alpha_{i,j} \leftarrow 1$; $p \leftarrow p + P_{i,j}$
12:             scheduled task k++
13:             $C \leftarrow C - s_i$
14:         else if $s_i > C$ then
15:             max partial size $l_i$
16:             for $j = 1; j \le l_i; j++$ do
17:                 if partial size $l_i < C$ then
18:                     $\alpha_{i,j} \leftarrow 1$
19:             $C \leftarrow C - l_i$; $n_i \leftarrow n_i - l_i$
20:     else
21:         break
22: maximum k, profit P
23: return $\mathcal{A} \leftarrow max\{\sum_{i=1}^{k} P_i, P_{k+1}\}$, profit p
```

The algorithm 1 greedily choose the most profit-effective tasks, which subject to network capacity and the nature of tasks. The approximation algorithm is proposed, and its time complexity and approximation ratio are analyzed in the next. Our algorithm is least $\frac{1}{2}\mathcal{OPT}$ in time bound $\mathcal{O}(nlog(n))$. Next, we will prove our approximation rate of algorithm is at least $\frac{1}{2}$.

Theorem 1 *Let $\mathcal{A}$ denote the set output by the algorithm. Then, $profit(\mathcal{A}) > \frac{1}{2}\mathcal{OPT}$.*

Proof: : Let $\mathcal{OPT}$ denote the optimal set. If $\sum_{i=1}^{n} s_i \le C$, then profit($\mathcal{A}$)= profit((OPT)). Therefore, we assume $\sum_{i=1}^{n} s_i > C$. We use k to denote the largest positive integer got by in algorithm line 3. We need to prove the following two inequalities

$$\sum_{i=1}^{k} P_i \le \mathcal{OPT} < \sum_{i=1}^{k+1} P_i \tag{8}$$

Obviously, the first inequality holds in our model. For the second inequality, in the algorithm line 1 we sort the tasks of task profit and task size which in the order of descend. The most efficient method is to schedule the most valueable tasks at firs until exceed the network capacity. The remaining capacity assigns to the most value flows until saturate the whole network. From the network perspective, $\sum_{i=1}^{k+1} P_i$ is the upper bound of profit (OPT). If we loose the constraint of flows completion number in tasks Equation (5) to smaller than n_i. We denote it as tasks completion indicator I_i and we use $0 < I_i < 1$ to replace with $I_i \in 0, 1$.

$$I_i = \begin{cases} 1 & i = 1, 2, \cdots, k, \\ \frac{C - \sum_{i=1}^{n} s_i}{s_{k+1}}, & i = k+1, \\ 0 & i = k+2, k+3, \cdots, n. \end{cases} \tag{9}$$

Here we get a maximum profit value $\hat{p}$ in linear program . Therefore, we conduct a inequality

$$\begin{aligned} \mathcal{OPT} \le \hat{p} &= \sum_{i=1}^{k} P_i + \frac{P_{k+1}}{s_{k+1}} \left(C - \sum_{i=1}^{k} s_i \right) \\ &< \sum_{i=1}^{k} P_i + \frac{P_{k+1}}{s_{k+1}} s_{k+1} = \sum_{i=1}^{k+1} P_i. \end{aligned} \tag{10}$$

In the end, we conduct following Equation

$$\mathcal{A} = max\{P_{k+1}, \sum_{i=1}^{k} P_i\} \ge \frac{1}{2} \sum_{i=1}^{k+1} P_i > \frac{\mathcal{OPT}}{2} \tag{11}$$

Accordingly, our algorithm approximation rate is at least $\frac{1}{2}$. ■

The algorithm 1 greedily choose the most profit-effective tasks, and the result of algorithm is acceptable and efficient.

C. Switch Design

SDN network [13] separates the network control plane from data forwarding plane which can improve resource utilization, network management and reduce cost. Controller has a full view of the whole network and flow information. According to the protocol of openflow [14], switches have flow tables when one unkown flow appears it trigger the packet-in and send the packet-in information to the controller. The controller computes the route and other bits in packet head and responses to switches. The pota switches use the simple mechanism on priority scheduling. Our switches only focus on the priorities of flows without considering task information. The limitation of Pota switches is it uses openflow switches and schedules flows based on the flow table. In specific, control has the information of deadline and profit which are not available in flow table and packtin message. The controller is responsible for computing priorities on both task-level and flow-level. In order to obtain maximum profit, the controller executes the algorithm 1 and schedules a set of flows need to be scheduled at current time slot. Then, the controller modifies the priorities of scheduling flows to absolute high priorities for authorities to scheduling in the switches. Switches resolve flows to transport based on priorities decided by the controller.

IV. Performance Evaluation

In this section we evaluate Pota performance in mininet. Our evaluation consists of four parts. First, we use mininet to build a topology to run our experiments. Second, we generate data mining and web search workload with deadline and profit information. Build on these, we realize the algorithm 1 on controller. In the end, we present the key result on task completion time, profit and task completion rate comparing with flow-level priority scheduling.

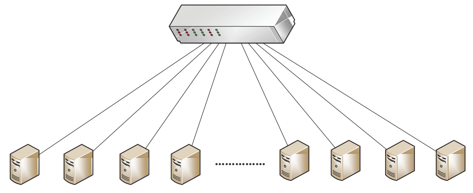


Figure 3. Topology used in simulations

A. *Experiment setting*

Workload: Our evaluation considers two representative workload web search and data mining. We generate workloads using python socket which send and receive flows according to Possion process based on characteristic of web search and data mining [10] [15]. We generate the flow size range from 1KB to 50MB and record them as trace file. In our evaluation, different workloads run on separate hosts in our mininet topology.

Topology: We use hose model topology in Figure 3 which can highlight the shortage of network resource. In our experiment, we could not scale up hosts number due to memory constrains, so we use hose model topology as 20 hosts connected to a single switch. Hose model topology has a severe limitation on switch capacity and congestion. We choose it to emphasize the significance of scheduling. Meanwhile, we emulate link bandwidth to 100Mbps.

Performance: We consider three performance benchmarks. Comparing to flow completion time, we use task completion time to illustrate the performance of our method. Since our method uses overall profit to improve task perform, we define a parameter of task completion rate to illustrate task performance. Profit is another quantity to directly measure task performance. In the meanwhile, we use the flow completion time of all the flows. We compare our method to priority schedule on task completion time, overall profit and task completion rate.

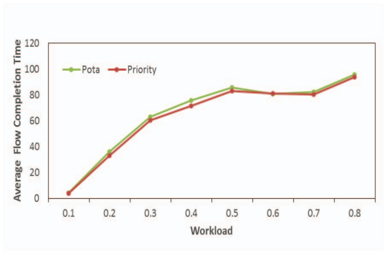
B. *Overall performance*

In this section we show Pota performance in star topology with simulation workloads. We show that Pota has a similar result to priority scheme with much higher profit and completion rate on task-level.

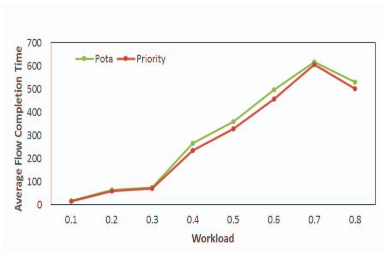
1) Task completion time: We first illustrate the completion time on task-level and flow-level. At first, we run one workload in the topology The web search workload have smooth and better result than data mining workload. It is because that the data mining workload flow size wider range from 1KB to 50MB, while the web search flow size only range from 1KB to 1MB. As shown in Figure 4, the flow completion time is normalize to best possible completion time (consume no congestion in network). In Figure 4, we bread down the FCT to four parts, separately describe the flow size variety. In Figure 4, Pota runs two kinds of workloads, the average flow completion time which is comparing with flow-level priority schedule. In Figure 4(c), we let 10 hosts run the web search workload and another 10 hosts run data mining workload. The routing process is not considered in our experiment. Pota computes the scheduling policy through trace and determines the the flows priorities. In this environment, mix workloads make Pota compute the profit and dynamic priorities leading a higher flow completion time. From Figure 4 it can be seen that our Pota is worse than priority scheduling but approximately in FCT. Flow completion time is a standard most scheduling methods applying, in Pota we introduce a standard of task completion time to illustrate our advantages. In task completion time(TCT) Pota is significantly better than priority scheduling showing Figure 5. Pota focuses on task-level scheduling, it relatively centralized schedules flows belong to one task, consequently it reduces TCT comparing to priority scheduling.

2) Task completion rate: Pota is a method aiming at improving task completion rate for maximum overall profit. Here we use task completion rate to be a standard to estimate the scheduling result. We regard the overall performance of scheduling methods as task completion rate. The deadline information is used to justify whether the tasks are completed. We run the mix workloads according to the trace file calculating the task completion rate shows in Figure 6 which clearly show the benefits from Pota in task completion rate. We observe that Pota sacrifice 8.1% flow-level latency for a incremental of 21% task completion rate which significant improves user experience and application performance. Pota unifies scheduling existing flows in tasks, as a result the task completion rate is better than flow-level priority scheduling methods.

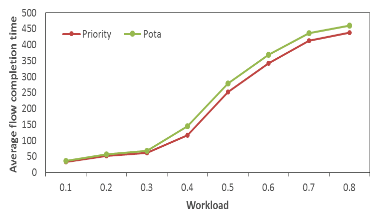
3) Task profit: Task profit is a definition we proposed in this paper as shown in Equation (1) and (2). We get the profit through successfully scheduling flows in tasks. The profit of web search and data mining task in our evaluation are normalized to integer for that web search workload has a profit of 2 and data mining workload has a profit of 1. In the algorithm 1, we can compute the maximum overall profit which is shown in Figure 7. In Figure 7, Pota significantly improves overall profit in the network. Flow-level priority scheduling method ignores the integrity



(a) Web search workload



(b) Data mining workload



(c) Mix workload

Figure 4. The average flow completion time of two workloads on web search and data mining and the mix workload

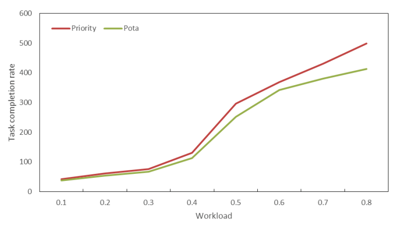


Figure 5. The task completion time

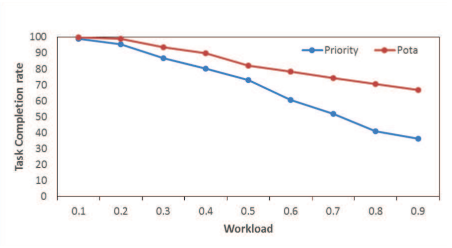


Figure 6. The task completion rate

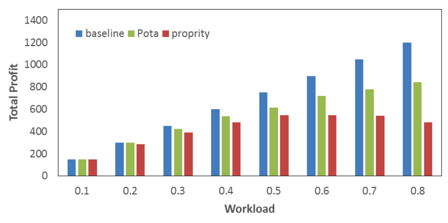


Figure 7. Profit comparison

of tasks. The differences between flow-level scheduling and Pota are that Pota guarantees the integrity of tasks and gets the total task profit, while flow-level scheduling method focuses on individual of flows without overall task profit.

V. Related work

Latency is critical for interactive data center applications. Many work in data center networks mainly focus on reducing latency or flow completion time. Specially, reduce latency and flow completion time are basically similar, and deadline is a special issue in latency. Recently some work recognizes importance of task and did some work on task-level scheduling. In this section, we briefly discuss some work relevant to Pota.

Hedera [6] shows elephant flows taking up bandwidth resource which cause short flows starving. HULL [16] makes effort to keep the queue small by using congestion control algorithm. HULL reserve some bandwidth for short flows and control packets in order to avoid elephant flow occupy all the bandwidth resources. Consequently, HULL obtain small latencies. Low network occupation can reduce latency, meanwhile, redundancy can also receive good result. In the endurable network condition, short flows redundancy [17] [18] can reduce completion times. In data center topology [19], fattree provides multipath between any two hosts that make replicate flows to minimize flow completion times possible. Reduce long tail flow completion time means the applications get better worst performance, DeTail [20] design to lower the long-tailed flow completion times on different data center workflows. L^2DCT [21] is a transport protocol to minimizing flow completion time by modulate the congestion window size based on estimate flow sizes.

DCTCP [1] keep the queue occupation under the threshold by introducing adaptive congestion control based on ECN. Low buffer utilization can partly reduce latency and miss deadline rate. D^3 [2]first proposed rate control scheme using deadline information, and assigns the rate by computing desire rate which use the information of flow size and flows deadline or estimated completion time. D^2TCP [3] is a deadline-aware TCP protocol extended by DCTCP. It implements to adjust the window size based on both deadline information and the congestion status. These methods mostly improved latency, however, they have limitations that they cannot precisely estimate the right flow rate to meet deadline when the early deadline flows arrive late. Moreover, bandwidth reservation and rate adjustment are hard to handle traffic bursty which is common in data center networks.

Having recognized above rate control strategy limitations, subsequent work pays attention to bandwidth preemption and the quality of responses. PDQ [4] and pFabric [5] are the state-of-the-art approaches in priority schedule. PDQ provides a distributed flow schedule mechanism based on assigning rates to flows using explicit feedback with flow priority. PFabric proposed a priority queue to schedule flows. Switches in pfabric just decide which flow to enqueue and choose the highest priority to dequeue. Task consist of multiple flows that for data center applications the integrity of one task is also important. Maximizing profit and guaranteeing the integrity of tasks are the main goals in our work.

Data centers have a large number of flows. All the flows in one task need to finish before deadline or task completion.

Considering this, some effort has been made in data center networks. Coflows [8] [9] proposed an application-level network abstraction and schedule problems. Coflow abstract the application into network semantic and use the semantic to optimize some application-level quality of service. On the basic of coflow, Varys [9] proposed a schedule problem to minimize flow completion time on rate allocation. On task-level scheduling, baraat [7] works on decentralized task-level schedule and flows in one task loosely synchronized by different pace. While we focus on centralize control making use of controller and increase application profits.

VI. Conclusion and Future work

Pota is a centralized method which is designed to complete tasks and meet their deadlines. It uses the deadline and task information to achieve maximum profit of scheduling tasks in data center networks. Tasks have a value of accomplishment, and flows in the tasks can dynamically effect the overall profit of the tasks on scheduling. Some urgent flows are more important comparing to normal flows. Pota has a character that it dynamically modifies priorities on flows according to time. And Pota has good adaptability for the nature of data center flows mixture. Our evaluation shows that the profit of task is a reasonable representation of the quality of service. Pota fits for both different and uniform flows, it significantly benefits the task completion rate, task completion time and overall profit comparing to existing priority scheduling methods.

We anticipate two directions of future work. First, the definition of profit in our paper is an abstract concept which is related to task priority and task value. Flows profit is determined by time and the task profit. Actually, both of completed flows number in one task and deadline emergency can effect the scheduling flows profit. We attempt to improve the model on profit to fit for practical setting. Second, we will further polish Pota and its approximation rate, and then implement and experiment it in a real data center network testbed environment.

Acknowledgment

This work is supported by the National Science Foundation for Distinguished Young Scholars of China (Grant No. 61225010); the State Key Program of National Natural Science of China(Grant No. 61432002); NSFC Grant Nos. 61272417, 61300189,61370199 and 61173161; Specialized Research Fund for the Doctoral Program of Higher Education (Grant No. 20130041110019), and the Fundamental Research Funds for the Central Universities (Grant. DUT15QY20).

References

[1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," *ACM SIGCOMM computer communication review*, 2011.

[2] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *ACM SIGCOMM Computer Communication Review*, 2011.

[3] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter tcp (d2tcp)," *ACM SIGCOMM Computer Communication Review*, 2012.

[4] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing flows quickly with preemptive scheduling," *ACM SIGCOMM Computer Communication Review*, 2012.

[5] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," in *SIGCOMM*, 2013.

[6] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *NSDI*, 2010.

[7] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized task-aware scheduling for data center networks," in *SIGCOMM*, 2014.

[8] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, 2012.

[9] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *SIGCOMM*, 2014.

[10] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *SIGCOMM*, 2009.

[11] H. El-Rewini, T. G. Lewis, and H. H. Ali, *Task scheduling in parallel and distributed systems*. Prentice-Hall, Inc., 1994.

[12] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," in *SIGCOMM*, 2013.

[13] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, 2008.

[14] O. S. Consortium *et al.*, "Openflow switch specification version 1.1. 0," 2011.

[15] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," *ACM SIGCOMM Computer Communication Review*, 2010.

[16] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is more: trading a little bandwidth for ultra-low latency in the data center," in *NSDI*, 2012.

[17] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Low latency via redundancy," in *Proceedings of the ACM conference on Emerging networking experiments and technologies*, 2013.

[18] H. Xu and B. Li, "Repflow: Minimizing flow completion times with replicated flows in data centers," in *INFOCOM*, 2014.

[19] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Computer Communication Review*, 2008.

[20] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "Detail: reducing the flow completion time tail in datacenter networks," *ACM SIGCOMM Computer Communication Review*, 2012.

[21] A. Munir, I. A. Qazi, Z. A. Uzmi, A. Mushtaq, S. N. Ismail, M. S. Iqbal, and B. Khan, "Minimizing flow completion times in data centers," in *INFOCOM*, 2013.