Optimal Cloud Instance Acquisition via IaaS Cloud Brokerage with Volume Discount

Ning Wang and Jie Wu
Center for Networked Computing, Temple University, USA
Email: {ning.wang, jiewu}@temple.edu

***Abstract*—Commercial cloud providers, e.g., Amazon EC2, offer the volume discount for large instance reservation in a time slot, and the majority of cloud jobs are delay-tolerant and do not need to be processed intermittently. These two features create an opportunity for the cloud brokerage service which aggregates and schedules cloud users' rental requests to earn volume discounts from cloud providers and sell to cloud users at a cheap price. A challenge for the broker is to properly schedule delay-tolerant jobs in order to maximize the volume discount amount over time. The scheduling idea is to generate several job bundles so each job bundle can get discount. In this paper, we discuss this problem from the homogeneous model first, where each job has the same processing time and delay-tolerant time, and we propose a dynamic programming approach. Then, we extend the model into the heterogeneous model, where the job processing time and the job deadline can be arbitrary values. In the heterogeneous scenario, we prove that the proposed problem is NP-hard even when the job processing time is unit. Then, we propose a greedy approach which turns out to have an approximation of $O(\ln n)$, where n is the total job number. Extensive trace-driven experiments from Google cluster trace demonstrates that our schemes achieve good performances.**

***Index Terms*—Cloud computing, Infrastructure-as-a-Service (IaaS), and scheduling.**

I. Introduction

Cloud computing is a large-scale distributed computing paradigm in which a pool of computing resources is available to users. Nowadays, the availability of high-capacity networks, low-cost computers, and storage devices have led to significant growth in cloud computing. Cloud computing eliminates the requirement for tenants, i.e., cloud users to plan ahead for provisioning, and allows enterprises to start from the small and increase resources only when there is a rise in service demand. Infrastructure-as-a-Service (IaaS) service model and visualization technologies are developed to provide resources to multiple users, and users do not need to worry about configuration. The users can specify the required software stack, e.g., operating systems and applications; they package them all together into Virtual Machines (VMs). There are several large-scale public cloud providers like Amazon EC2 [1], Windows Azure [2], and Google cloud platform [3].

As for the cloud rental pricing, cloud providers usually adopt an hourly billing scheme even if the customers do not actually utilize the allocated resources in the whole billing slot. The pricing model can be briefly divided into two types: the pay-as-you-use pricing model and the reservation model. In former model, users pay cloud usage on-demand at a high price. In the later model, the user can pre-pay at the discounted price to reserve cloud resource. The drawback is that it is hard to pre-estimate the usage, and thus there might exist waste. In addition, to stimulate cloud users usage in the current market, many cloud providers provide volume discount, i.e., the user will get an extra discount when they reach a certain cloud usage. Amazon EC2 [4] and Telecoms Cloud [5] use a three-tiered volume discount model. Rackspace [6] provides a two-tiered volume discount model. Azure [7] also has the volume discount, but the detailed model is available upon request.

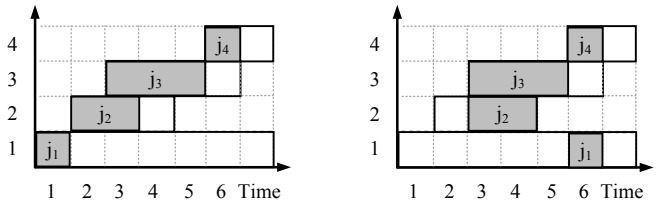


Fig. 1. An illustration of scheduling with volume discount.

TABLE I
Comparison of different matchmaking strategies.

	Time		
	arrival	process	deadline
job 1	1	1	7
job 2	2	2	3
job 3	3	3	6
job 4	6	1	7

Due to the complexity of billing strategies in different cloud providers, the demand for Cloud Services Brokerage (CSB) appears. A cloud services brokerage is a third party company that adds value to cloud services on behalf of cloud service consumers. The cloud service brokerage market size is expected to grow from USD 4.50 Billion in 2016 to USD 9.52 Billion by 2021 at an estimated Compound Annual Growth Rate (CAGR) of 16.2% [8]. Some famous CSBs are IBM Cloud Brokerage [9] and Appirio [10]. The current business model of CSB includes aggregation and integration. However, the business model for cloud brokerage is still evolving. Recent works [11, 12] found that a cloud broker can help reduce the cost of customers through temporal multiplexing of resources. By temporal multiplexing, the broker takes advantage of providers volume discount and save the user's cost.

In this paper, we focus on the volume discount optimization in cloud brokerage service. An illustration of the network model

978–1–5386–2542–2/18/$31.00

is shown in Fig. 1, where there are four non-preemptive jobs in total. The job arrival time, processing time, and the deadline information are shown in Table I. If there is no broker service, each user will schedule the job upon arrival as shown in Fig. 1(a). In the toy example, if we apply a simple two-tiered volume discount policy, i.e., if the number of rental instances at a particular time slot reaches two, the cloud rental gets discounted, and Fig. 1(a) gets 2 discounted instances at time slot 3. Through proper cloud scheduling, we can get a better rental result as shown in Fig. 1(b). In Fig. 1(b), this schedule gets six discounted instances at time slots $3, 4$, and 6.

In this paper, we address the optimal non-preemptive job schedule problem for cloud brokerage service. Specifically, cloud broker receives a series of cloud jobs and knows the corresponding processing time and the finishing deadline for them. The cloud broker would like to minimize the total cloud rental cost of all jobs by taking advantage of volume discount under the deadline constraint. Therefore, each user pays less and users are encouraged to submit more jobs to the cloud. We formulate the problem as a Bundle Job Scheduling (BJS) problem and discuss the volume discount function in a general situation, i.e., a concave function.

We discuss the BJS problem in two scenarios. In the homogeneous model, where each job has the same processing time and delay-tolerant level, we propose a dynamic programming approach to find the optimal solution. Then, we extend the model into a general heterogeneous case, where each job's processing time and delay-tolerant level can be arbitrary values. In the heterogeneous setting, the BJS problem turns out to be NP-hard even when the job processing time is united. Then, we propose a series of greedy algorithms and one of the greedy approaches is proved to have an approximation ratio of $O(\ln n)$, where n is the number of jobs.

The contributions of this paper are summarized as follows:

- To our best knowledge, we are the first to consider this optimal non-preemptive job scheduling with discount in cloud scheduling.
- In the homogeneous case, we propose a dynamic programming approach to find the optimal solution.
- In the general heterogeneous case, we prove that the BJS problem is NP-hard, and a heuristic algorithm with an approximation ratio of $O(\ln n)$ is proposed.
- We verify the effectiveness of proposed approaches in Google cluster data trace.

The remainder of the paper is organized as follows: The related works are in Section II. The problem statement is introduced in Section III. The corresponding dynamic programming algorithm in the homogeneous model is provided in Section IV. The NP-hardness of the BJS problem in the general case and an approximation algorithm is presented in Section V. The experimental results from real cloud traces are shown in Section VII, and we conclude the paper in Section VIII.

II. Related Works

In this section, we briefly summarize the related works and issues in cloud scheduling related problems.

TABLE II
AMAZON CLOUD VOLUME DISCOUNT STRATEGY [4]

Total Reserved Instances	Upfront Discount	Hourly Discount
Less than $ 500, 000	0%	0%
$ 500,000 to $ 4,000,000	5%	5 %
$ 4,000,000 to $ 10,000,000	10%	10%
$ More than $ 10,000,000	special discount	special discount

1) Different Rental Approaches: In [13], the authors considered the optimal cloud rental problem under three different reservation methods, i.e., on-demand rental, scheduled reserved rental, and 1-year reserved rental, where the rental cost decreases at the cost of rental flexibility loss. A greedy solution is proposed with offline demand. In [14], the authors only considered two different reservation methods, on-demand and reserved rental. However, they moved one step further by proposing an online approach with a 2-competitive ratio. The common assumption in these two papers is that jobs have to be scheduled right away upon arrival. Therefore, there is no scheduling issue.

2) Volume Discount in Cloud Services: The volume discount is widely used in commercial cloud providers. Amazon EC2 [4], Telecoms Cloud [5], and Rackspace [6] use the tiered discount model. The current tiered model in Amazon EC2 is shown in Table II. Other companies do have volume discount, but they do not provide the detailed strategy, such as Azure [7]. In [15], they used a tiered discount policy for each VM and formulated the cloud rental problem as a coalition formation game among a set of cloud service. Users consider their heterogeneous computing demands, rational selfishness, and non-transferable utility to make the decision. In [11], the authors proposed a general concave pricing model. The difference between their job model and this paper is that the job is non-preemptive in this paper, and improper scheduling may lead to a bad influence on the future schedule.

3) Cloud Scheduling: The idea behind cloud scheduling [16–20] is that the job might not be scheduled right away upon arrival. It is reasonable to postpone some time under the deadline constraint to reduce the cloud rental cost. The main idea is the same as energy efficient scheduling. That is, we would like to aggregate jobs into certain time slots which have a high workload and stop at the other time slots. In [17], they considered cloud rental reorganization problem to increase the workload for each rental servers by shutting down some servers with low utilization. However, there is a reorganization cost. For energy efficient scheduling, this scheduling can reduce the CPU energy cost. In [18, 19], authors discuss the speed scaling technique under the preemptive job and non-preemptive job model, respectively. In [20], authors consider the bandwidth scaling in the data center network.

In this paper, we focus on the cloud scheduling problem to maximize the volume discount for users. The different rental approaches are not discussed in this paper, but the proposed approach can be applied to different rental approaches.

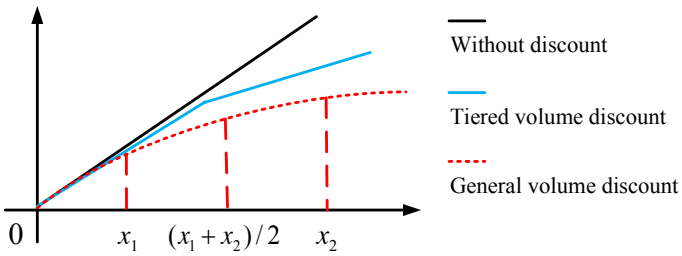


Fig. 2. An illustration of volume discount in the pricing strategies: there are three pricing strategies, no volume discount, general concave pricing model, and the common tiered pricing model

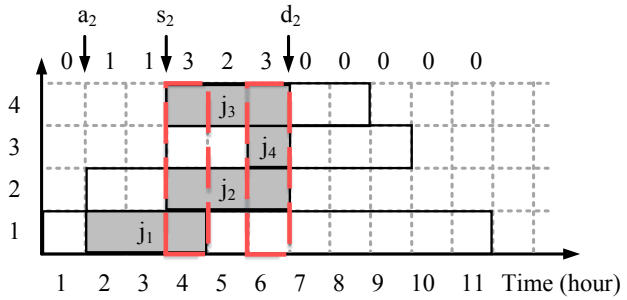


Fig. 3. An illustration of job model and there is a two-tiered discount model, when volume in a time slot reaches 3, there is a volume discount.

III. Problem Statement

In this section, we first discuss the volume discount model and job model in cloud rental followed by the problem formulation and discussion.

A. Cloud Rental Model

The commercial cloud providers are using a slot-based pricing model, e.g., an hour or a day, even if the customers do not actually utilize the allocated resources in the whole billing slot. To simulate cloud users, many cloud providers, such as Amazon EC2 [4], Telecoms Cloud [5], Rackspace [6], and Azure [7] provide volume discount, i.e., cloud users can rent cloud at a discounted rate when they reach a certain usage. Though different cloud service providers may offer different pricing strategies for volume discount, a pricing strategy $C(\cdot)$ with volume discount can be modelled as a non-decreasing concave function in general. There are two requirements for any x_1, x_2, with $x_1 \leq x_2$:

- $C(x_1) \leq C(x_2)$,
- $C(x_2) - C(\frac{x_1+x_2}{2}) \leq C(\frac{x_1+x_2}{2}) - C(x_1)$,

where the first requirement ensures that the more resource that you rent, the more money that you need pay. This is the basic reasonable requirement for the cloud rental, since there is no reason to get more resource with less money. The second requirement ensures that the more you rent, the less unit price that you will get. Otherwise, the volume discount is meaningless. An illustration of volume discount is shown in Fig. 2. The tiered pricing model is a special case of the concave function. Note that we focus on volume discount caused by a large number of instance rental. Note that there are different cloud reservation methods, e.g., pay-as-you-go and reserved pricing, the detailed cloud reservation method is out of this paper's scope, and the volume discount can be applied into any reservation method.

B. Job Model

In this paper, we assume that there are n delay-tolerant jobs. The arrival times of job requests are arbitrary, and the processing time for each job is deterministic and known to the broker based on estimation. Therefore, for each job, we use a three-tuple $j_i : \{a_i, p_i, d_i\}$ to uniquely model it, where a_i is the job arrival time, p_i is the processing time, and d_i is the job deadline. Note that $d_i - a_i \geq p_i$, and the extra time provides a certain scheduling flexibility. The range $[a_i, d_i]$ is called flexibility range in the following of this paper. We assume the time can be discretized into slots according to the billing cycles, and each job arrives and finishes at the beginning of a time slot. This is reasonable, since in the real computation, the cloud scheduler checks available jobs and schedule them in a slotted manner. In any unit time slot, a job either is allocated with no resource or uses allocated resource in the whole time slot. Therefore, there is a corresponding starting time, s_i, and finishing time, f_i for job i, $f_i = s_i + p_i$. Note that we assume that each job consumes the same workload to the cloud, e.g., a VM, and we consider the non-preemptive job processing model, which means that job is executed until completion. In the middle, the job processing cannot be interrupted.

C. Cloud Brokerage Service

If jobs are tolerant to a certain scheduling latency, which is the common case, the cloud users can benefit from this mediation. The cloud brokerage service provides an intermediate layer for cloud users and mediates the job requests in a manner which benefits the most from the volume discounts provided by the cloud provider and meets the job deadline at the same time. The cloud provider benefits from the revenue boosted by the brokerage. For the delay non-tolerant job, the cloud broker can still pull multiple cloud services into a single user interface and utilize different charging strategies to save user's rental cost. In this paper, we focus on the extra volume discount with cloud brokerage scheduling.

An illustration of the job model and volume discount is shown in Fig. 3, where there are 4 jobs in total, i.e., $j_1 : \{1, 3, 11\}, j_2 : \{2, 3, 6\}, j_3 : \{4, 3, 8\}$, and $j_4 : \{6, 1, 9\}$, respectively. The scheduled starting times of jobs j_1 to j_4 are $2, 4, 4$, and 6, respectively. Based on this assignment, we can calculate the volume in each time slot, shown at the top of the Fig. 3. Let us use a simple two-tiered price model in this example, i.e., a discount price is granted when the volume in a time slot is no smaller than three. As a result, when $t = 4$ and 6, we get a discounted cloud rental price. The total workload with discount is 6. However, if we always schedule these four jobs upon their arrival, you will find that there is no discounted rental at all, which demonstrates the potential benefit of cloud scheduling.

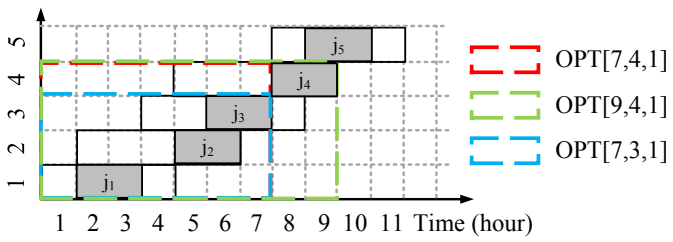


Fig. 4. An illustration of dynamic programming in the homogeneous model, where each job has the same processing time, scheduling flexibility but different arrival times.

D. Problem Formulation

Based on the proposed job and cloud rental models, we propose the Bundle Job Scheduling (BJS) problem for cloud brokerage service, which can be mathematically formulated into the following format:

$$\begin{aligned} \min \quad & \sum_{t=1}^{T} C(X_t) \\ \text{s.t.} \quad & \sum_{t=a_i}^{d_i-p_i} x_t^i = 1, \quad x_t^i = \{0,1\}, \quad \forall i, \\ & X_t = \sum_{i=1}^{n} \sum_{t'=t-p_i}^{t} x_{t'}^i, \quad \forall i, t, \\ & a_i \le x_t^i \le d_i - p_i, \quad \forall i, \end{aligned} \tag{1}$$

where t denotes the time slot. For each feasible assignment of job i, A time slot or several time slots will be determined to process the job and formulate a set. We use a decision value, x_t^i to denote the assignment decision of job i. That is, $x_t^i = 1$ means that job i is scheduled to start processing at time slot t. Otherwise, $x_t^i = 0$ means that job i does not start at time slot t. Therefore, the first constraint of Eq. 1 means that each job can only be scheduled once. The second constraint of Eq. 1 refers to the accumulated number of jobs at a time t and we use X_t to denote it. The third constraint is the decision feasibility constraint. The objective of Eq. 1 is the overall cloud rental cost of a scheduling, where $C(\cdot)$ is the volume-cost function.

E. Problem Discussion

To our best knowledge, the proposed BJS problem is unique due to volume discount pricing. The proposed problem seems to be similar to the machine scheduling problems [21, 22]. In machine scheduling problems, the targets are always to minimize the makespan of all jobs, average latency of total jobs, or the total tardiness, the total latency of finishing processing after the due time. The major difference between the BJS problem and machine scheduling problems is that machine scheduling problems always want to schedule the jobs as soon as possible and latency is important. However, as for the BJS problem, it does not optimize the scheduling latency but schedules jobs into a series of bundles to maximize the total volume discount.

(a) before swap (b) after swap

Fig. 5. An illustration of optimal substructure in homogeneous job model.

IV. Homogeneous Job Model

In this section, we first discuss the BJS problem in the homogeneous case, where all jobs have the same job processing time and flexibility range, i.e., $d_i - a_i = d_j - a_j, \forall i, j$. In the homogeneous job model, the BJS problem can be optimally solved by dynamic programming technique.

An illustration of the network model in the homogeneous model is shown in Fig. 4, where the feasibility range is shown as the dotted rectangle area. The actual scheduling is denoted as a dark rectangle. First, jobs are sorted according to their arrival times. Without loss of generality, let us assume that if $i < j$, job i arrives no later than job j. Note that in this case, we use p to denote the processing time of any job without loss of generality.

Theorem 1. *In the homogeneous job model, for two jobs i and j ($i < j$), job i should be always scheduled first than job j without losing the optimality.*

Proof. If the accumulated job is too few or too many in every time slot, the scheduling of jobs i and j are not important, since in the former case, there is no discount at all. In the latter case, the volume discount is already maximized. In the following of the proof, we focus on the case, where adding one or two jobs can further improve the volume discount in some time slots. The total range where jobs i, j can influence can be divided into three different time ranges according to jobs i, j's arrival time and deadline, that is, $[s_i, s_j], [s_j, d_i]$, and $[d_i, d_j]$, respectively. We will discuss the results of two schedule strategies in Figs. 5(a) and 5(b), to check if the proposed scheduling strategy can increase the overall discounted workload maximally and we will prove this case by case.

For the range $[s_i, s_j]$, since job j's arrival time is later than that of job i's arrival time and same as the deadline, job j cannot be scheduled at that range. Therefore, if adding one more job can increase the accumulated volume discount, we should schedule job i first into range $[s_i, s_j]$. If $f_i \le s_j$, scheduling job i first does not generate more volume discount in $[s_j, f_i]$. However, if $f_i > s_j$, after job i scheduled, the workload of time slots in $[s_j, f_i]$ increases, which may lead to a better schedule decision for job j. Therefore, job i should be scheduled first. If adding one more job can increase the accumulated volume discount most in the range $[s_j, d_i]$, we can use jobs i and j to cover this range with no difference. However, if we jointly consider the second-most discount workload increase in ranges $[s_i, s_j]$ or $[d_i, d_j]$, the job i (job j) should be scheduled in $[s_i, s_j]([d_i, d_j])$. In each case, job i will be scheduled first. If adding one more job can increase the accumulated volume discount in range $[d_i, d_j]$, we can prove that changing the

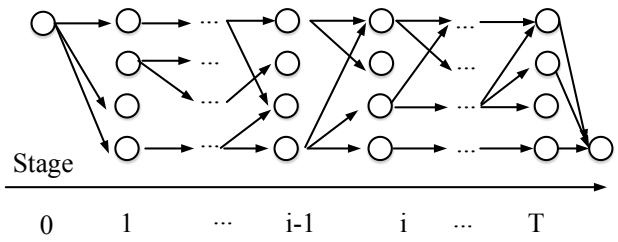


Fig. 6. An illustration of dynamic programming in a general case.

scheduling sequence can improve the possible volume discount in the same way, since the schedule in 5(b) can increase the volume discount in range $[d_i, d_j]$. □

Based on the Theorem 1, we can sort all jobs based on their arrival times. Without lose of generality, we can denote jobs based on their arrival time, i.e., the j_1 is the first arrived job. After that, we gradually find the optimal solution up to time slot i with first j jobs, as shown in Fig. 4. To simplify the following description, we define the overlapping set as follows: for each job, its overlapping set is a set of previous jobs which can be scheduled together with the current job. For example, j_3 's overlapping set is $O_3 = \{j_1, j_2, j_3\}$, j_4's overlapping set is $O_4 = \{j_2, j_3, j_4, j_5\}$, and j_5 's overlapping set is $O_5 = \{j_3, j_4, j_5\}$, respectively.

We will discuss the detailed dynamic programming technique in the BJS problem. To make the dynamic programming procedure easy to follow, we gradually discuss the dynamic programming solution from the most simple case, i.e., united processing time, to the general case, i.e., arbitrary processing time. Let us denote the state, the optimal solution up to time slot i and first j jobs as $OPT[i, j]$. Initially, all states are ∞, except $OPT[0, 0]$, which equals to 0. If the workload of each job is 1,

$$OPT[i, j] = \min_{j' \in O_i} \{OPT[i-1, j'-1] + C[j', j]\}, \quad (2)$$

where $C[j', j]$ is the cost to schedule jobs from j' to j together. Note that when j' equals to j, it represents that the job j is scheduled alone. Therefore, all the intermediate jobs can be scheduled as well, i.e., all the jobs from j' to j are scheduled at time slot i. There is no combination issue in such case, and Eq. 2 can be applied into different cost models.

Then, we extend to the case where the processing time of each job is 2 time slots. We define the state $OPT[i, j, k]$, which is the optimal solution upto time slot i, ends up with the job j. This means that the job j finishes at time slot i, and the number of jobs finished at time slot i is k. Therefore, the optimal state updating procedure is as follows:

$$OPT[i, j, k] = \min\{OPT[i', j-k-k', k'] + C[j-k-k', j]\} \quad (3)$$

The i and i' represent the finishing time of jobs i and i'. The i' can be any time within p of the time slot i. Similar, the k' determines the workload situation at time slot i'. The k is up bounded by the size of job j's overlapping set.

Algorithm 1 Dynamic Programming Algorithm

Input: Job information and rental cost function

Output: The scheduling result X.

```
Sort the jobs according to the arrival time.
Initialize OPT[i, j, ···] = ∞ except OPT[0, 0, ···] = 0, c_min = ∞.
for Each time slot i with first j jobs do
    Find job j's overlapping job set.
    for All job j's overlapping job set do
        Calculate the optimal state updating as Eq. 5.
        if OPT[i, n, k_{n-p+2}, ··· , k_i] < c_min then
            Update X.
```

Let us calculate the optimal states, when we add j_5, as shown in Fig. 4. The smallest i is 9 in this example, otherwise, we cannot find a feasible schedule for j_5. In this case, i' can be $i-1$ and $i-2$, that is, i' can be 8 or 7. The j' can be any jobs that have overlap with feasible range with the job j. In Fig. 4, j' can be $3, 4$, and 5, where $j' = 3$ means that jobs $3, 4$, and 5 are scheduled together. A state updating example is shown below,

$$\begin{aligned} OPT[9,5,1] = \min\{&OPT[8,4,0] + C[5,5], \\ &OPT[7,4,0] + C[5,5], OPT[7,4,1] + C[5,5] \\ &OPT[7,4,2] + C[5,5], OPT[7,3]\} \\ OPT[9,5,2] = \min\{&OPT[8,3,0] + C[4,5], \\ &OPT[8,3,1] + C[4,5]\} \end{aligned} \quad (4)$$

In Eq. 4, we ignore the infeasible states in the data update. The feasibility can be check in $O(n)$. In addition, we only consider the perfect overlapping in the cost calculation, since the non-perfect overlapping case can be regarded as that we only add one job each time.

Based on the aforementioned discussion, we can extend it to a general case where the job length can be arbitrary p. Let us denote the state in this case is $OPT[i, j, k_{i-p+2}, \cdots, k_i]$, where $\{k_{i-p}, k_{i-p+1}, \cdots, k_i\}$ is the workload at that left p time slots before current time slot i. Therefore,

$$\begin{aligned} &OPT[i, j, k_{i-p+2}, \cdots, k_i] \\ &= \min\{OPT[i', j-k_i-k'_{i-p+1}, \cdots, k'_i] \\ &+ C[j-k_i-k'_i, j]\}, \end{aligned} \quad (5)$$

where the intermediate state is updated by reducing the last $k_i + k'_i$ jobs. Though we can solve the problem in the general case as shown in Fig. 6, the dynamic programming presented above is computationally intractable. This is because calculating all states results in exponential time complexity. Also, to store all these states, the space complexity is exponential upto $O(q^{p+1})$, where q is the size of maximum overlapping set size. This is known as the curse of dimensionality suffered by all high-dimensional dynamic programming. To update the Eq. 5, one has to compute for all states. However, since a state space is

defined in a high-dimensional spacerecall that it is defined as a $p+1$-tuple, there exists exponentially many such states. Each job has different processing time, the time complexity further increases due to the feasibility checking.

V. Heterogeneous Job Model

In this section, we discuss the solution in the general case, where jobs are heterogeneous, i.e., each job has an arbitrary arrival time, processing time, and deadline, and the BJS problem is proved to be NP-hard in the general heterogeneous case. Therefore, we propose two heuristic approaches, and the theoretical analysis is also provided.

A. NP-hardness proof

Theorem 2. *The proposed BJS problem is NP-hard in heterogeneous model.*

Proof. The proposed BJS problem is a combinatorial optimization. We prove the BJS problem is NP-hard by first proving that the BJS problem belongs to NP class because that for a given scheduling, H, we can verify if all constraints are satisfied simultaneously in polynomial time. Now we show its NP-hard by a reduction of the weighted mutually exclusive maximum set cover [23].

The weighted mutually exclusive maximum set cover problem is as follows: given a ground set U of n elements, a collection S of m subsets of U, we try to find a sub-collection S' of S with the minimum number of subsets such that (1) no two subsets in S' are overlapped and (2) the number of elements of the union of all subsets in S' is the maximum. If we assign each subset in F a weight (a real number) and further require that the weight of S', i.e. the weighted sum of subsets in S' is minimized.

The reduction from a special case of BJS problem to the mutually exclusive maximum set cover problem is as follows: In a special case of BJS problem, all jobs have the same weight and can be finished in one time-slot. For each time slot, each possible schedule is a set, whose weight is schedule cost. Clearly, each job cannot be scheduled multiple times, which is the mutually exclusive requirement in the weighted mutually exclusive maximum set cover problem. or each time slot, there is a set for each job itself, we can always assign it S' without avoiding constraint (1). Therefore, the maximum number of the union of all subsets in S' is the size of U. □

The insight of the proof is that in general case jobs have different feasible processing ranges. A job's schedule might have an influence on some jobs which arrives in the distant future, and its influence cannot be limited, such as Theorem 1 in Section IV. An example is shown in Fig. 3, where job j_2's schedule can influence the job j_4's schedule. Furthermore, different jobs schedule orders lead to different results like jobs j_2 and j_3, and it is hard to control the optimality. In this section, we propose the following greedy approach, which has an approximation bound.

To solve the BJS problem, we first propose a heuristic solution, called Most-Overlap (MO) Algorithm. The idea is that

Algorithm 2 Most Overlapping (MO) Algorithm

Input: Job information and rental cost function

Output: The scheduling result X.

1: **for** i from 1 to T **do**
2: Calculate the feasible jobs set s.
3: $|\mathbb{S}|$ is the unassigned jobs, $s^\star$ is the current maximum candidate set.
4: **while** $|\mathbb{S}| > 0$ **do**
5: initialize $|s^\star| = 0$.
6: **for** i from 1 to T **do**
7: **if** $|s_i| > |s^\star|$ **then**
8: $s^\star = s_i$.
9: Remove $s^\star$ in each time slot's candidate set.
10: Update X and $|\mathbb{S}|$.

Algorithm 3 Most Efficient (ME) Algorithm

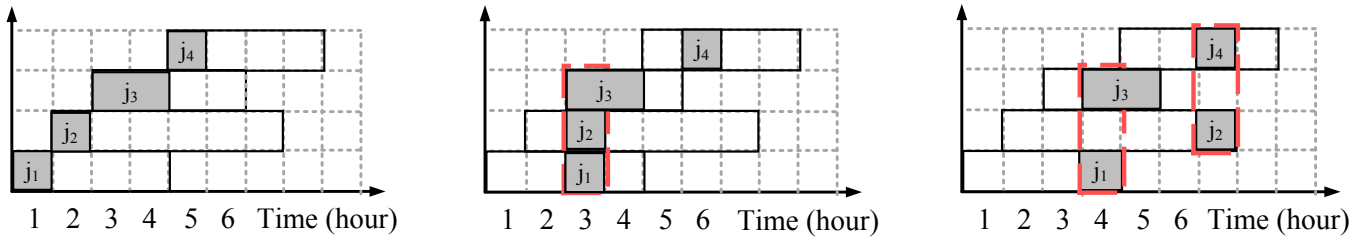
Input: Job information and rental cost function

Output: The scheduling result X.

1: **for** i from 1 to T **do**
2: Calculate the feasible jobs set s.
3: $|\mathbb{S}|$ is the unassigned jobs, $s^\star$ is the current maximum candidate set.
4: **while** $|\mathbb{S}| > 0$ **do**
5: $\theta_{min} = \infty$, $s^\star = \phi$
6: **for** i from 1 to T **do**
7: Find all combination in s_i with a total number of N.
8: **for** j from 1 to N **do**
9: Calculate the cost by assigning each combination at i as c_i.
10: **if** $c_i / \sum_{j_i \in s_i} |j_i| \leq \theta_{min}$ **then**
11: $\theta_{min} = c_i / \sum_{j_i \in s_i} |j_i|$, $s^\star = s_i$
12: Remove $s^\star$ in each time slot's candidate set.
13: Update X and $|\mathbb{S}|$.

we iterate the whole time slots to find a time slot, which has the most number of feasible jobs and schedule job at that time. Once we find that time slot, we schedule all feasible jobs in that time slot. We keep doing until all jobs are scheduled. The MO Algorithm is shown in Algorithm 2. In line 4, while there is an unassigned job, the MO algorithm will iterate all time slots as in line 6. In line 7, if we find a time slot which can schedule more unassigned jobs than the current best solution, the MO algorithm will update the current solution.

In Fig. 7(b), the MO algorithm finds the time slot 3 at the first round, since it has the maximum size of 3. Therefore, the job j_1, j_2, j_3 will be scheduled at time slot 3. After that, job j_4 is the only left, and it will be scheduled at one of its feasible time. One drawback of MO algorithm is that scheduling two many jobs in one time slot will reduce the volume discount opportunity for later arrival jobs. In Fig. 7(b), if job j_2 or job j_3 is scheduled later together with job j_4, we can further increase the volume discount. Another drawback is due to the heterogeneous job



(a) First-fit algorithm (b) MO algorithm (c) ME algorithm

Fig. 7. An illustration of proposed algorithms, where there is a two-tiered volume discount model, when the volume reaches 2 in a time slot.

length, a good overlapping in the current time slot may not be a good overlapping in the following time slot. This problem becomes significant when some job has a long processing time, In Fig. 7(b), only a half of job j_3 get volume discount. The time complexity of the MO algorithm is $O(nT)$, since in the worst case, MO algorithm will have n rounds and each round it will check T times.

A nature improvement of MO algorithm is to schedule all jobs based on their overlapping situation. Specifically, each job tries all its possible assignments and finds the time slots which overlap with other jobs' flexibility range most. Therefore, we do not only consider the overlapping situation for a job in a time slot but the overlapping situation for a job in its flexibility range. Here, we propose the idea of scheduling set, where multiple jobs can be scheduled at the same time in a scheduling set. First, we calculate all possible schedule combinations in a time slot. If multiple jobs' schedule overlaps at a particular time, there is a set whose elements are the set of jobs and its cost/weight is the corresponding cloud rental cost. Note that if a job's schedule does not overlap with other jobs, there is a set which has only one element.

The idea of the Most-Efficient (ME) Algorithm approach is that we calculate all feasible job scheduling combination at each time slot and evaluate jobs greedily based on average cost. The idea of the greedy algorithm is that we give each set a price θ, the price is the normalized cost of the set, i.e., $\frac{c_i}{|s|}$. After that each time we select the most cost-effective set in the remaining set and update the corresponding job schedule. In line 4, while there is an unassigned job, the ME algorithm will iterate all time slots as in line 6. In line 7, we will calculate all the combination in that slot. In lines 8 to 11, we find a set of jobs which can add to the current scheduling in a most-efficient manner.

An example to illustrate the ME algorithm is shown in Fig. 7(c). In this example, we assume that there is a 2-tiered pricing model, where the original cost for a unit workload is 1, and if the workload of a time slot is larger or equal to 2, the cost for a unit workload is 0.5. We calculate the possible job combination for each time slot. At the round 1, the job set $\{j_2, j_4\}$ at time slot 7 will be scheduled first due to the highest cost-efficient ratio, i.e., $0.5 \times 2/2 = 0.5$. At the round 2, since the job set $\{j_1, j_3\}$ is the lowest average cost in this example, i.e., $(0.5 \times 2 + 1 \times 1)/3 = 0.67$. From Fig. 7(c), the ME algorithm overcomes the drawback of the MO algorithm, scheduling two many jobs in the same time slot by considering all combination of jobs. Therefore, there is a performance-complexity trade-off. The time complexity of ME algorithm is $O(n2^nT)$ in the worst case, where the MO algorithm will have n rounds at most, and each round it will check T times. Each checking can be $O(2^n)$ in the worst case. However, considering that the jobs do not arrive at the same time, the real performance of ME will be much better.

Theorem 3. *The proposed ME algorithm achieves a* $\ln n$ *approximation ratio in unit length.*

Proof. For explanation convenience, let us assume that jobs are selected according to the order of $j_1, j_2, \cdots, j_n$, the cost of optimal solution is OPT, and the set of jobs schedules are $\{s_1^\star, s_2^\star, \cdots, s_l^\star\}$, $s_i^\star \in S$. Let us assume that $I := \{1 \le i \le l | s_i^\star \cap \bar{X} \neq \phi\}$, which are available sets that can be added to solution in the optimal solution but not in the greedy solution. According to the definition of OPT and greedy character of ME algorithm,

$$\begin{aligned} OPT &= \sum_{1\le i\le l} c(s_i^\star) \ge \sum_{i\in l} |s_i^\star - X| \frac{c(s_i^\star)}{|s_i^\star - X|} \\ &= \sum_{i\in L} |s_i^\star - X|\theta(j_k) \ge |\bar{X}|\theta(j_k)| \\ &\ge (n-k+1)\theta(j_k), \end{aligned} \tag{6}$$

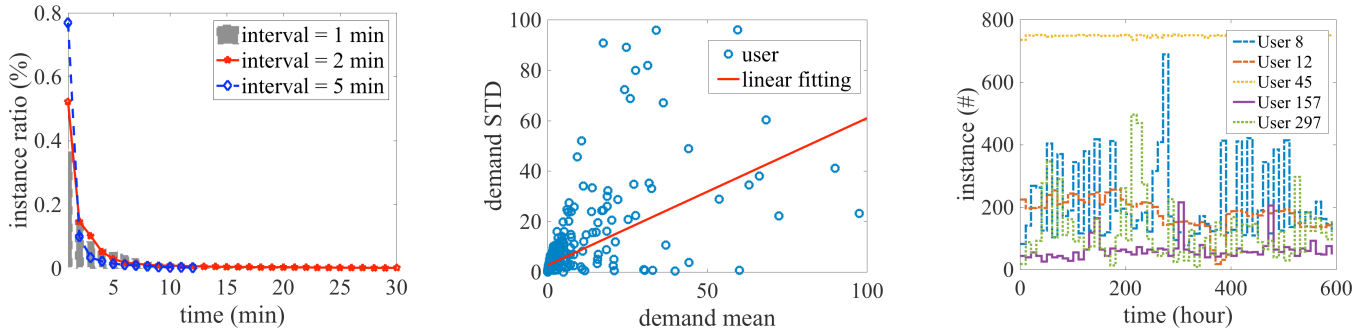
where $\theta(j_k)$ is the amortized cost for each unit of workload of j_k. The first and the second inequality in Eq. 6 is due to the fact that $s_i^\star$ may have overlap with the result of the ME algorithm, X. The last inequality is due to the fact that each time at least one job will be added to the solution set.

Since the overall cost of ME approach is $\sum_{1\le i\le n} \theta(j_i)$, based on the Eq. 6,

$$\begin{aligned} \sum_{1\le i\le n} \theta(j_i) &= \sum_{s\in X} \sum_{t\in \mathbb{S}-X} \theta(j_i) \\ &\le \sum_{1\le i\le n} \frac{1}{i} OPT = \ln n OPT \end{aligned} \tag{7}$$

Therefore, we prove that the ME algorithm has an approximation ratio of $\ln n$ in united length. □

When the job length is more than 1, due to the property of concave rental cost function, the BJS problem has the submodular property, and thus there is a $O(\ln n)$ approximation ratio for the ME algorithm based on [24].



(a) Job length distribution (b) Demand distribution (c) Sample user demand

Fig. 8. An illustration of job information in Google cluster dataset.

VI. Performance Evaluation

In this section, we demonstrate the effectiveness of proposed algorithms by using the real Internet trace.

A. Trace Introduction

In the second set of experiments, we conduct simulations based on Google cluster data [25], which has been widely used to perform cloud computing related experiments [11, 14]. This data contains a large number of job records coming from 933 users. It is recorded for a 29-day duration in May 2011 on a cluster of 11k physical machines. The comprised data size is over 40GB. In the experiment, we consider the jobs with explicit computational jobs which can be processed in a deterministic duration given the precise processing power of the machine used. To align with our problem, we preprocessed the data to eliminate the jobs that arrive before the trace collection and jobs that are not naturally finished. This leaves us with $385,359$ job records coming from 389 users. In Google cluster data, the job event status, submit, schedule, evict, fail, finish, etc, is provided for each job. The processing length of each job can be calculated through job event status. The job requests submitted by different users exhibit different patterns in term of job length and demand arrival distribution. Times in usage measurements are treated slightly differently because the maximum measurement length is 300 seconds. We apply the same start-time offset to these times as we do for events inside the trace window; this is enough to ensure a clear separation between events before the trace and other events.

B. Experiment Setting

We conduct experiments by using the following setting: the arrival time of a job is the timestamp when we find a job status 0. The job length is calculated based on time difference between the scheduling starting time and job finish time, which can also be retrieved from the job status. Note that the time is discretized to different levels in the experiments, specifically, we use $[1, 5, 10]$ minutes. We refer the method from [11] to generate the deadline, which is,

$$d_i = p_i \cdot (1 + \rho) + a_i \tag{8}$$

where we choose ρ from $\{1, 5, 10, 20\}$. The value ρ represents the flexibility level of job scheduling. In order to reduce the computational complexity, we arbitrarily choose 5 to 20 users and schedule their jobs in experiments. As for the cloud cost rental model, we propose three different setting, the two-tiered model, three-tiered model, and a concave function, $f(x) = \sqrt{x}$. We conduct three different settings in the experiments: (1) the number of users, (2) the different ρ value, and (3) the different time discretized levels. We refer the pricing scheme of Amazon and Rackspace to get a three-tiered volume discount model and two-tiered volume discount model, respectively. Note that in the Google cluster trace, we do not have that many tasks in a particular time: therefore, we reduce the discount threshold accordingly in the experiments. The experiments in each setting are repeated for 100 times.

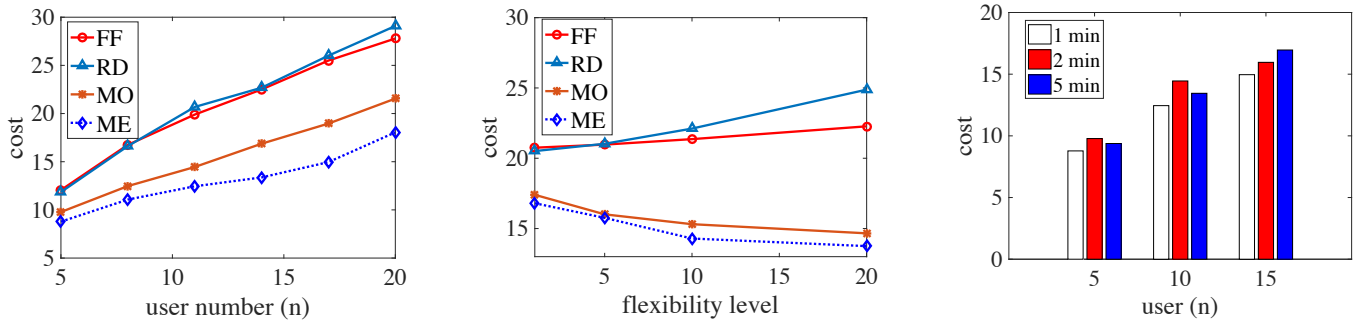
C. Algorithm Comparison

We compare the performance of the proposed algorithm under different topology settings. (1) Most-Overlapping (MO) algorithm, which we explained in Section V. (2) Most Efficient (ME) algorithm, which we explained in Section V. (3) First-fit (FF) algorithm, which simply assigns jobs when they arrive and is the current cloud scheduling method. (4) Random (RD) algorithm, which randomly selects a matching for a user. RD algorithm is a baseline approach.

D. Results Analysis

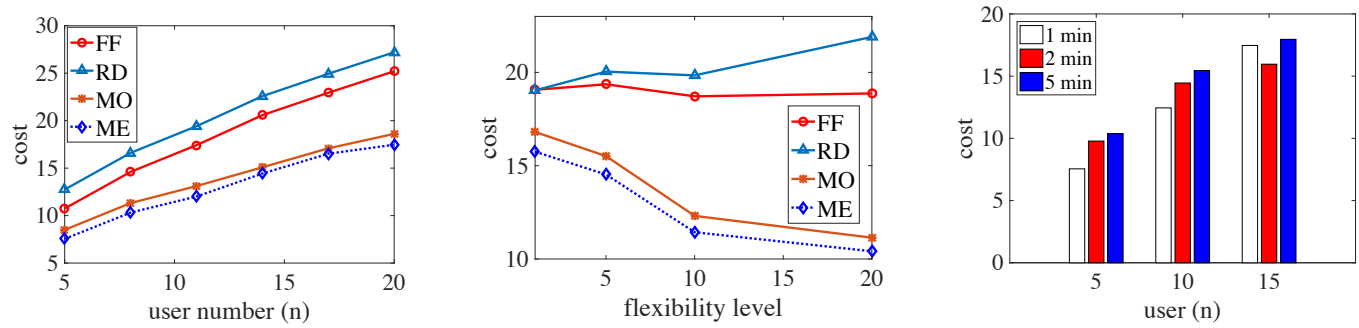
Fig. 8 presents the trace analysis results from Google cluster dataset. Specifically, Fig. 8(a) show the job length distribution follows the exponential distribution, and the majority of job can be finished within 10 minutes. In Fig. 8(a), the job's length is rounded based on 1 min, 2 min, and 5 min. As for the job arrival pattern, the results are shown in Figs. 8(b) and 8(c). Fig. 8(b) shows the job arrival is not uniform, and it is not practical to use the period scheduling scheme for cloud rental request optimization. Fig. 8(c) shows that the cloud request pattern of five random users in 600 hours. The results show that the user demand pattern is devised and thus demonstrates the necessity of cloud scheduling to utilize the different demand pattern for different users to earn volume discount.

Fig. 9 gives the performance results where a two-tiered pricing model is applied in the different setting. In Figs. 9(a) and 9(b), we can find that the ME algorithm always achieves the lowest cloud rental cost, followed by the MO algorithm, the FF and RD algorithms have a similar performance. Particularly, when the amount of cloud user increases, the number of



(a) $\rho = 2$ (b) $n = 10$ (c) $\rho = 2$

Fig. 9. The performance result in two-tiered pricing model.



(a) $\rho = 2$ (b) $n = 10$ (c) $\rho = 2$

Fig. 10. The performance result in three-tiered pricing model.

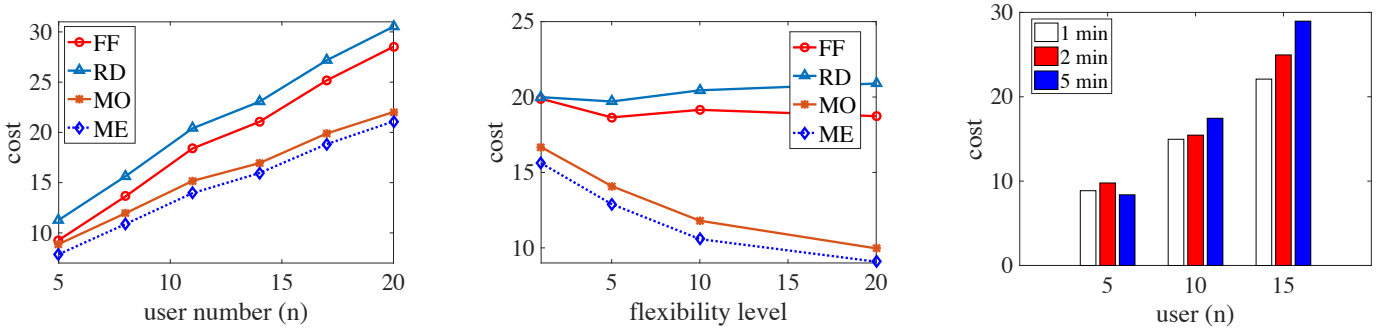
cloud rental cost increase for all four algorithms. The bad performance of FF and RD algorithms demonstrate that always scheduling job upon arrival or scheduling job randomly has a low chance to take advantage of volume discount and thus leads to the high rental cost. The MO algorithm achieves a much better performance compared with FF and RD algorithms by scheduling jobs together. However, since the MO algorithm considers the limited number of job combination, its performance is worse than the ME algorithm, the ME algorithm reduces the cost about a half compared with the RD algorithm, and 20% cost compared with the MO algorithm at the cost of higher complexity.

Fig. 9(b) shows that along with the flexibility level increase, the cloud rental cost for FF and RD algorithm increases because there is a smaller chance for jobs to overlap. The random property of the RR algorithm leads to the highest cost. The FF algorithm's performance is better due to the uneven task arrival distribution. These two algorithms do not take advantage of the job scheduling flexibility. For MO and ME algorithms, they can utilize the flexibility to reduce the cloud rental cost. In Fig. 9(b), the MO and ME reduce about 20% and 25% cloud rental cost on average, respectively, compared with the the FF algorithm. Since in the experiments, we discrete time into time slots and we would like to check the different discretized schemes. Fig. 9(c) show that the coarse-grained discretized scheme has the higher cost due to inefficient overlapping information.

Fig. 10 show the performance results of four algorithms in a three-tiered pricing model. The results in Figs. 10(a), 10(b), and 11(a) show that the proposed ME algorithm always leads to the lowest cost in different settings. The MO algorithm has the very similar performance compared with the ME algorithm, i.e., 5% higher cost, but it has a lower time complexity. Therefore, there is a trade-off in the algorithm selection in real applications. When the flexibility level is high, i.e., $\rho = 20$, the ME and MO algorithm roughly reduce a half of the overall rental cost. One interesting result is that if we compare the performance in Figs. 9(b) and 10(b), we can find that in the three-tiered volume discount pricing model, the FF and RD algorithms cannot take advantage of it at all, and their performances remain the same. However, for the MO and ME algorithms, they further reduce the cost in multiple-tiered volume discount model. In Fig. 9(b), when $\rho = 20$, the cost is reduced by 17% on average. In Fig. 10(b), when $\rho = 20$, the cost is reduced by 36% on average.

Fig. 11 shows the performance results of four algorithms in a general concave function. The result is similar to the three-tiered cost model as shown in Fig. 11(a). The MO algorithm achieves a closer performance compared with the ME algorithm in the general concave model than its performance in the 2-tiered model may due to the reason that in 2-tiered model, there is no volume discount at all for time slots which do not reach the threshold. However, in the general concave function, it becomes easier to get some volume discount, even the discount is limited. From Fig. 11(b), we can get the result that the rental cost decreases with the flexibility level increase.

Figs. 9(c), 10(c), and 11(c) show the performance results of the ME algorithm in different setting in terms of different length granularity, and we find that the coarse-grained discretized scheme has an influence about the scheduling accuracy. The ME algorithm achieves the worst performance in the coarse-grained discretization. Another result is that the fine-grained discretization improves more performance when we have a general concave pricing model.



(a) $\rho = 2$ (b) $n = 10$ (c) $\rho = 2$

Fig. 11. The performance result in $\sqrt{x}$ pricing model.

VII. Conclusion

In this paper, we consider a cloud brokerage service schedule optimization by considering the volume discount in cloud rental the discount by reserving a large pool of instances in a time slot. Considering the fact that there is schedule flexibility, a job does not need to be scheduled at its arrival but within the deadline. Therefore, a problem is to properly schedule jobs in order to maximize the price discount amount. The idea behind the schedule is to generate several job bundles so that each job bundle has a discounted price. In this paper, we propose a general non-decreasing concave pricing model. Then, we discuss this problem from the homogeneous model, where each job has the same schedule range, finishing deadline minus arrival time and propose a dynamic programming approach. Then, we extend the model into a general case: the proposed problem turns out to be NP-hard even when the job processing time is unit. Then, we propose a greedy approach which turns out to be $O(\ln n)$, where n is the number of jobs. Extensive trace-driven experiments from Google cluster traces demonstrates that our schemes achieve good performances.

VIII. Acknowledgement

This research was supported in part by NSF grants CNS 1757533, CNS1629746, CNS 1564128, CNS 1449860, CNS 1461932, CNS 1460971, and IIP 1439672.

References

[1] https://aws.amazon.com/ec2/
[2] https://azure.microsoft.com/
[3] https://cloud.google.com/
[4] "Amazon ec2 pricing." https://aws.amazon.com/ec2/pricing/reserved-instances/pricing/
[5] "Telecoms cloud volume discount." https://www.telecomscloud.com/volume-discounts/
[6] "Rackspace pricing." https://www.rackspace.com/en-us/cloud/servers/discounts
[7] "Microsoft azure pricing." https://azure.microsoft.com/en-us/community/partners/get-started/
[8] "Cloud services brokerage market." https://www.marketsandmarkets.com/Market-Reports/cloud-brokerage-market-771.html
[9] "Ibm cloud brokerage." https://www.ibm.com/us-en/marketplace/cloud-brokerage-cam/resources#product-header-top
[10] "Appirio cloud brokerage." https://appirio.com/
[11] R. Zhang, K. Wu, M. Li, and J. Wang, "Online resource scheduling under concave pricing for cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 4, pp. 1131–1145, 2016.
[12] W. Wang, D. Niu, B. Li, and B. Liang, "Dynamic cloud resource reservation via cloud brokerage," in *Proceedings of the IEEE ICDCS*, 2013.
[13] Q. Wang, M. M. Tan, X. Tang, and W. Cai, "Minimizing cost in iaas clouds via scheduled instance reservation," in *Proceedings of the IEEE ICDCS*, 2017.
[14] W. Wang, B. Li, and B. Liang, "To reserve or not to reserve: Optimal online multi-instance acquisition in iaas clouds." in *Proceedings of the USENIX ICAC*, 2013, pp. 13–22.
[15] J. Wang, X. Xiao, J. Wang, K. Lu, X. Deng, and A. A. Gumaste, "When group-buying meets cloud computing," in *Proceedings of the IEEE INFOCOM*, 2016.
[16] N. Wang and J. Wu, "Maximizing the user's benefit in the mobile cloud computing," in *Proceedings of the ACM MobiCom S3*, 2016.
[17] C. Tian, Y. Wang, Y. Luo, H. Jiang, W. Liu, J. Wu, and H. Yin, "Minimizing content reorganization and tolerating imperfect workload prediction for cloud-based video-on-demand services," *IEEE Transactions on Services Computing*, vol. 9, no. 6, pp. 926–939, 2016.
[18] L. L. Andrew, A. Wierman, and A. Tang, "Optimal speed scaling under arbitrary power functions," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 2, pp. 39–41, 2009.
[19] A. Antoniadis and C.-C. Huang, "Non-preemptive speed scaling," *Journal of Scheduling*, vol. 16, no. 4, pp. 385–394, 2013.
[20] D. Li, C. Chen, J. Guan, Y. Zhang, J. Zhu, and R. Yu, "Dcloud: deadline-aware resource allocation for cloud computing jobs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 8, pp. 2248–2260, 2016.
[21] J. K. Lenstra, A. R. Kan, and P. Brucker, "Complexity of machine scheduling problems," vol. 1, pp. 343–362, 1977.
[22] E. Mokotoff, "Parallel machine scheduling problems: A survey," *Asia-Pacific Journal of Operational Research*, vol. 18, no. 2, p. 193, 2001.
[23] S. Lu, G. Mandava, G. Yan, and X. Lu, "An exact algorithm for finding cancer driver somatic genome alterations: the weighted mutually exclusive maximum set cover problem," *Algorithms for Molecular Biology*, vol. 11, no. 1, p. 11, 2016.
[24] P.-J. Wan, D.-Z. Du, P. Pardalos, and W. Wu, "Greedy approximations for minimum submodular cover with submodular cost," *Computational Optimization and Applications*, vol. 45, no. 2, pp. 463–474, 2010.
[25] J. Wilkes, "More Google cluster data," Google research blog, Nov. 2011, posted at http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html.