Minimizing Cost in IaaS Clouds via Scheduled Instance Reservation

Qiushi Wang*, Ming Ming Tan†, Xueyan Tang†, Wentong Cai†
*Multi-plAtform Game Innovation Centre †School of Computer Science and Engineering
Nanyang Technological University, Singapore
Email: {qswang, mmtan, asxytang, aswtcai}@ntu.edu.sg

***Abstract*—Regular diurnal patterns are often seen in the workloads of cloud-based online applications. This kind of non-stationary workloads changes the processing demands over time. To run application services with minimum costs, the number of cloud instances can be dynamically adjusted according to the workload variations. Recently, a new type of scheduled instances has emerged in the Infrastructure-as-a-Service market to facilitate such configurations. Scheduled instances can be reserved based on a recurring schedule and they offer price discounts. Meanwhile, cloud vendors require minimum scheduled durations to avoid the overhead of frequently launching and terminating cloud instances. Coupled with traditional on-demand and reserved instances, it becomes more complicated for users to find the optimal combination of these three pricing options to minimize their monetary costs. For the new scheduled instances, not only the number of instances but also their start and stop times have to be decided. In this paper, we develop a fast and effective strategy to solve this problem. Based on the hourly workload distributions, we first compute the optimal number of instances to acquire for each pricing option. Then, we design a scheduling algorithm to arrange the scheduled instances in compliance with the restriction of their scheduled durations. Using the workloads of the LOL online game and the Wikipedia Mobile service as two case studies, the efficacy of our strategy is demonstrated.**

I. Introduction

The growth of Infrastructure-as-a-Service (IaaS) is accelerating and attracting rapidly increasing attention. According to a latest report [1], the worldwide public cloud services market will grow by 17.2% to a total of $208.6 billion in 2016. The highest growth comes from IaaS, which is projected to grow by 42.8%. Some analysts have pointed out that the world is shifting workloads to clouds, and IaaS vendors may eventually replace legacy hardware vendors [2].

IaaS enables customers to acquire processing capacities on demand to meet their time-varying computation requirements. In addition to high scalability and fast deployment, the growth of IaaS is also stimulated by that monetary costs can be saved as an outcome of public cloud adoption [3].

Normally, IaaS vendors offer diverse pricing options to satisfy the requests of cloud users. With the changes of market conditions, the price policy is adjusted to benefit all market participants. Taking an IaaS market leader – Amazon as an example, Table I lists the latest price policy of a general-purpose Amazon EC2 instance. Comparing Table I with an old price policy of Amazon shown in Table II [4], a new option of renting the instances by scheduled reservation has emerged. The new scheduled instances enable users to purchase capacity reservations that recur on a daily, weekly, or monthly basis, with a specified start time and duration, for a one-year term [5]. Figure 1 shows the web interface for purchasing scheduled instances. This new pricing option provides a more flexible choice for periodical workloads that repeat on regular schedules. Besides the new scheduled instance, the price policy for long term reservation has also changed. Previously, users can reserve a long-term instance (reserved instance) with an upfront fee and enjoy an hourly usage price discount. But in the latest policy, the one-time upfront fee has been raised and the hourly usage charge has been cancelled. That means regardless of whether the reserved instance is running or not, users will be charged the same fee. These two price changes reflect that IaaS vendors prefer regular and steady demands.

The new price policy makes it more complicated for cloud users to purchase instances to meet their workload demands with minimum monetary costs. Previous studies [4], [6], [7] have shown that running the entire workload under a single pricing option is usually not cost-effective. With a time-varying workload, users have to decide how many instances should be long-term reserved to optimize cost. As

Table I
Pricing options of an m4.large instance (Linux, Current Generation, US East) in Amazon EC2, As of Oct.1, 2016

Instance Type	Pricing Option	Upfront	Hourly
m4.large	On-Demand	$ 0	$ 0.12
	1-Year Reserved	$ **603**	$ **0**
	Scheduled Reserved *	$0	$ 0.114 [a]
			$ 0.108 [b]

* The total scheduled duration should be greater than 1200 hours/year, 100 hours/month, 24 hours/week, or 4 hours/day.
[a] Peak hours (Monday 0:00 to Saturday 0:00) 5% off On-Demand usage charges
[b] Off-Peak hours (Saturday 0:00 to Monday 0:00) 10% off On-Demand usage charges

Table II
Pricing of Instances (Light Utilization, Linux, US East) in Amazon EC2, As of Feb.10, 2013

Instance Type	Pricing Option	Upfront	Hourly
Standard Small	On-Demand	$ 0	$ 0.08
	1-Year Reserved	$ 69	$ 0.039
Standard Medium	On-Demand	$0	$ 0.16
	1-Year Reserved	$138	$ 0.078

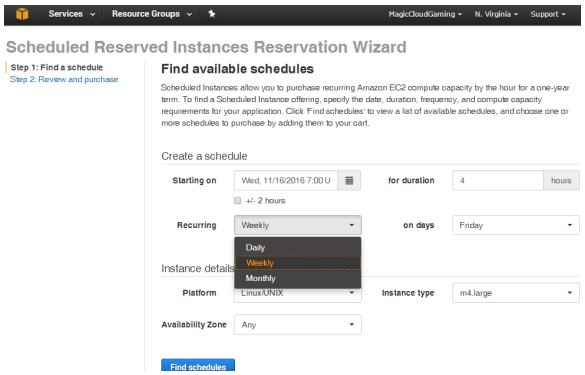


Figure 1. Web interface for reserving scheduled instances

the new scheduled instance comes into the market, users can further make use of it for regular workloads and save more cost. For example, in the scenario of popular online games, high workloads often come regularly in every evening. If long-term reserved instances are purchased for the peak workloads, a large amount of upfront fee needs to be paid, but quite a lot of computation resources will be wasted during low gaming times. On the other hand, completely switching to on-demand instances may also incur excessive cost due to their high hourly usage prices. To cope with regularly fluctuating workloads, adopting scheduled instances can yield a better trade-off in this dilemma. However, the existing strategies proposed for choosing between traditional on-demand and reserved instances [6], [8], [9] cannot deal with scheduled instances directly. As the scheduled instance is designed particularly for a planned demand, instance allocation strategies must be adapted to take into account the regular workload variations.

In this paper, we propose a fast and effective strategy to solve the cost optimization problem with the aforementioned three pricing options. First, we decouple the cost optimization problem into two parts. The first sub-problem is to decide how many scheduled instances and reserved instances should be booked respectively. For scheduled instances, the numbers of instances to book at different hours in a day are normally not identical when the workload shows a strong diurnal pattern. These numbers are identified through the hourly workload distributions. For reserved instances, a searching method is proposed to find the optimal allocation solution by progressively replacing scheduled/on-demand instances with reserved instances. The second sub-problem is to arrange the scheduled instances in compliance with the restriction of their scheduled durations. As shown in Table I, for each scheduled instance purchased, its running time must be longer than 4 hours per day, etc. We propose a scheduling algorithm to package the workload demands into scheduled instances so that all of them are arranged to run for sufficient time and satisfy the duration requirement.

We take two case studies: an online game League of Legend (LOL) and the Wikipedia Mobile service. Through an official API [10], we crawled the workload data of the LOL game for two years. The workload data of Wikipedia Mobile service is obtained from the statistics publicly available [11]. We analyze the workloads of these two popular online applications. Our analysis shows that the workloads of both applications are highly variable and the workload variation shows a strong regular pattern which fits well with the features of the new scheduled instance. We evaluate our cost optimization algorithms with these two workloads.

The remainder of this paper is organized as follows. Section II introduces the new type of scheduled instances in detail. Section III presents our proposed algorithms to solve the cost optimization problem. Section IV uses two real workloads as case studies to demonstrate the efficacy of our algorithms. We briefly survey the related work in Section V. Finally, Section VI concludes the paper.

II. Scheduled Instance

Before introducing the new type of scheduled instances, we briefly review the pricing details of the traditional on-demand and reserved instances. On-demand instances allow users to pay for the resources based on the actual usage time. Let p_{od} denote the hourly price of an on-demand instance. If an on-demand instance is run for h hours, then its cost is $p_{od} \cdot h$. Reserved instances require users to prepay an upfront fee C_{re} for a long-term commitment (one year) and then the usage is free. So, the cost of a reserved instance is simply the one-time upfront fee. From the workload perspective, reserved instances better fit for long-term stable workloads, whereas on-demand instances better match with highly variable workloads. It is intuitive that the optimal choice between an on-demand instance and a reserved instance is determined by the instance utilization. Suppose that the average utilization of an instance over one year is α $(0 \leq \alpha \leq 1)$. Then, the cost of using an on-demand instance is $C_{od} = p_{od} \cdot \alpha \cdot T$, where $T = 24 \times 365 = 8760$ is the number of hours in a year. If $C_{od} < C_{re}$ or equivalently $\alpha < \frac{C_{re}}{p_{od} \cdot T}$, using an on-demand instance is more economical. Otherwise, a reserved instance is preferred.

The new scheduled instances provide a better option for periodical workloads that repeat regularly. We use an example to illustrate the advantage of scheduled instances. Consider a daily workload pattern shown in Figure 2(a), which demands two instances from 13:00 to 16:00, and one instance from 8:00 to 13:00 and from 16:00 to 22:00. If no scheduled instances are available, we should purchase a reserved instance for the long term and use an additional on-demand instance from 13:00 to 16:00 everyday. According to the prices shown in Table I, the total cost of one year is $603 + 0.12 \times 3 \times 365 = \$734.4$. On the other hand, with scheduled instances, we can reserve two scheduled instances as shown in Figure 2(b), where the first instance runs from 8:00 to 16:00 and the second instance runs from 13:00 to 22:00 everyday. The total running hours of the two instances are 17 hours. Thus, the total cost is

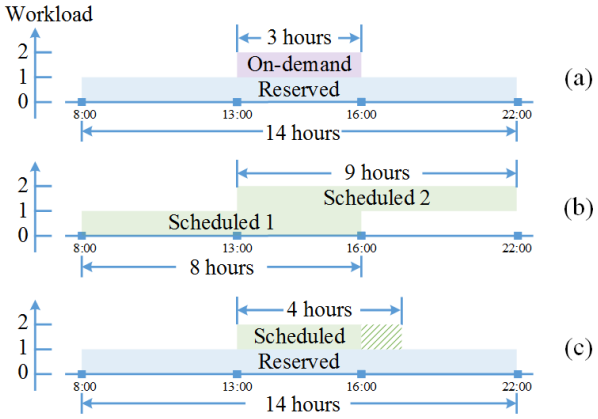


Figure 2. Advantage and challenge of scheduled instances

$0.114 \times 17 \times 261 + 0.108 \times 17 \times 104 = \$696.7$ (assume there are 261 weekdays and 104 weekend days in a year). It can be seen that using scheduled instances is cheaper than using traditional reserved and on-demand instances.

Finding the optimal combination of the three pricing options to minimize cost is a challenging problem. Although the hourly cost of a scheduled instance lies between those of on-demand and reserved instances, we cannot simply replace some on-demand instances in the best combination of two traditional pricing options with scheduled instances. In the above example, the on-demand instance in Figure 2(a) is used for only 3 hours per day, which is less than the minimum duration required for a scheduled instance (4 hours, see Table I). If we directly replace the on-demand instance with a scheduled instance running for 4 hours as shown in Figure 2(c), the total cost would be $603 + 0.114 \times 4 \times 261 + 0.108 \times 4 \times 104 = \$766.9$, which is even higher than using only reserved and on-demand instances. This example shows that determining the optimal choice of scheduled instances is not a simple problem. To satisfy the duration requirement of scheduled instances, the start and stop times of the two instances have to be rearranged as shown in Figure 2(b), which also affects the number of reserved instances to purchase. Thus, the three pricing options should be considered as a whole in the cost optimization.

III. Cost Optimization Strategy

The workloads of many online applications are known to have diurnal patterns, e.g. online games [12], [13] and web services [14], [15]. To take advantage of the respective features of different pricing options, a smart strategy is to serve stable base workloads with reserved instances, to serve periodical workloads with scheduled instances, and to serve the remaining abrupt peak workloads with on-demand instances. To develop an efficient solution, we decompose the cost optimization problem into two parts. First, we decide the number of instances to acquire for each pricing option, and then, we design a schedule for each scheduled instance under its duration restriction.

A. Instance Allocation Algorithm

We start by developing an instance allocation algorithm to find the optimal number of instances to acquire for each pricing option, without considering the duration restriction of scheduled instances. We address the duration restriction later when determining the start and stop times of scheduled instances. As introduced earlier, scheduled instances can be reserved with repeated patterns recurring daily, weekly or monthly. In this paper, we shall focus on reserving scheduled instances with daily recurring patterns. Our methods can easily be extended to reserve scheduled instances with weekly or monthly patterns.

The instance allocation algorithm consists of two steps. First, we identify the maximum possible numbers of scheduled instances needed for each hour of a day, assuming that reserved instances are not available. Then, we progressively replace the scheduled/on-demand instances by the reserved instances until the total cost cannot be further reduced. This gives the optimal combination of the three pricing options.

Since scheduled instances can only be started and stopped on the hour, we model the computation demands of an application by hourly workload distributions. For each hour i of a day, let $n_i(x)$ be the residual probability distribution of the number of instances needed for that hour. That is, $n_i(x)$ is the probability for the number of instances needed to be at least x in hour i. Figure 3 shows two sample distributions for the 9th and 22nd hours of a day which are drawn from the workload of a real online game to be introduced in Section IV.

Given the hourly workload distributions, we can easily determine the optimal number of scheduled instances to acquire for that hour, assuming that reserved instances are not available. Specifically, if the discounted price of scheduled instances is a fraction α_s of the price of on-demand instances, then the optimal number for that hour is given by $B_i = n_i^{-1}(\alpha_s)$, where $n_i^{-1}(\cdot)$ is the inverse function of $n_i(\cdot)$, i.e., $n_i(B_i) = \alpha_s$.[1] If more than B_i scheduled instances are reserved for the hour, some scheduled instances would have utilization lower than α_s and thus replacing them with on-demand instances can save cost. On the other hand, if less than B_i scheduled instances are reserved, some on-demand instances to acquire for that hour can have utilization higher than α_s and hence replacing them with scheduled instances can save cost. Therefore, the optimal number of scheduled instances to reserve is B_i. In the example of Figure 3, suppose that $\alpha_s = 95\%$. Then, as shown by the red dashed lines, for the 9th hour, $B_9 = n_9^{-1}(95\%) = 322$ scheduled instances should be used, and for the 22nd hour,

[1] Amazon EC2 currently has different price discount rates on weekdays and weekend days (Table I). In this case, we can compute an average discount rate (with weights of 5 and 2 for weekdays and weekend days) for the purpose of determining the optimal number of scheduled instances to reserve.

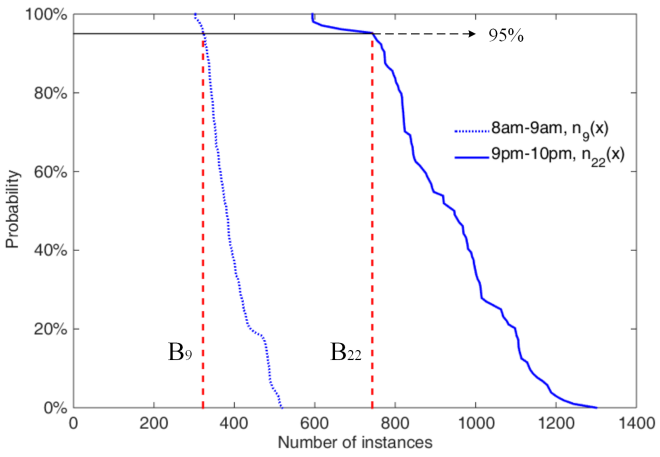


Figure 3. Samples of hourly workload distributions

$B_{22} = n_{22}^{-1}(95\%) = 743$ scheduled instances should be used.

Note that the distributions $n_i(x)$ describe the number of instances needed for each hour. Conceptually, if we index all the instances and always assign the workload to the lowest-indexed instances, then the utilization of the instances would decrease with their indexes. So, another way to read Figure 3 is to consider each x-axis value as an instance index. Then, each point (x, y) implies that the instance indexed x has a utilization of y. The numbers B_i computed above essentially provision each instance with a combination of scheduled instances and on-demand instances. Specifically, for the instance indexed x, in each hour i of a day, if $B_i \geq x$, a scheduled instance is acquired to serve its workload in hour i. Otherwise, if $B_i < x$, an on-demand instance is acquired to serve its workload in hour i. In the absence of reserved instances, this is the best way to provision the instance indexed x, and the total cost of the instance over one year is given by $c(x) = 365 \cdot (\sum_{i,B_i \geq x} \alpha_s \cdot p_{od} + \sum_{i,B_i < x} n_i(x) \cdot p_{od})$, where p_{od} is the hourly price of an on-demand instance, α_s is the price ratio between a scheduled instance and an on-demand instance, and 365 is the number of days in a year. In the example of Figure 3, consider the instance indexed 400. Since $B_9 = 322 < 400$ and $B_{22} = 743 > 400$, an on-demand instance is acquired to serve as instance $\#400$ in the 9th hour and a scheduled instance is acquired to serve as instance $\#400$ in the 22nd hour.

Now, let us further take the reserved instances into consideration. With the reserved instances available, each indexed instance x can either be reserved for one year so that no scheduled or on-demand instance needs to be acquired to serve its workload, or be provisioned with the above best combination of scheduled instances and on-demand instances. To minimize cost, we should compare the costs of these two options and choose the option with lower cost. Let C_{re} be the one-time upfront fee of a reserved instance. If $c(x) > C_{re}$, a reserved instance should be used to provision the indexed instance x; if $c(x) \leq C_{re}$, a combination of scheduled instances and on-demand instances should be used

Instance Allocation Algorithm:
Input: hourly workload distributions $n_i(x)$
Output: R - the number of reserved instances to acquire
r_i - the number of scheduled instances to acquire in each hour i

for $i = 1$ to 24 **do**
  $B_i = n_i^{-1}(\alpha_s)$;
$x = 0$;
do let $x = x + 1$;
  compute $c(x)$;
while $c(x) \geq C_{re}$;
$R = x - 1$;
for $i = 1$ to 24 **do**
  $r_i = \max\{0, B_i - R\}$;
return R and all r_i;

Figure 4. Instance allocation algorithm

to provision the indexed instance x. Thus, to find the optimal combination of the three pricing options, we can compute the cost of each indexed instance x based on the numbers B_i obtained earlier and check the cost against C_{re} to decide whether to convert the indexed instance x into a reserved instance. In the case that it is converted into a reserved instance, the number of scheduled instances (if any) to acquire for each hour should be reduced by 1 accordingly. Figure 4 shows the pseudo code of the instance allocation algorithm. Since the instances have decreasing utilizations and hence decreasing costs with their indexes, the checking can stop once an indexed instance with cost lower than C_{re} is encountered. To speed up, a binary search can also be used to find the optimal number of reserved instances to purchase.

B. Scheduling Algorithm

The instance allocation algorithm in the previous section outputs a demand list $r_1, r_2, \cdots, r_{24}$ for scheduled instances for all the hours of a day, where r_i represents the number of scheduled instances needed in hour i. In this section, we develop an algorithm to determine a reservation schedule of scheduled instances to meet the demands in a given list. The main challenge is to deal with the minimum duration requirement of scheduled instances. Recall that once a scheduled instance is started, it has to run continuously for a minimum duration of 4 hours. Hence, it will be desirable to find a reservation schedule such that the entire duration of every scheduled instance is fully utilized to serve the demands (i.e., no part of its reservation period is left idle). As analyzed below, this may not always be feasible.

To facilitate presentation, a scheduled instance is said to be *active* from the time it is started till the time it is stopped. Note that scheduled instances can only be started and stopped on the hour. If an instance is started at hour s and stopped at hour t, it is active during hours $s, s+1, s+2, \cdots, t-1$. A demand list is said to be *feasible*

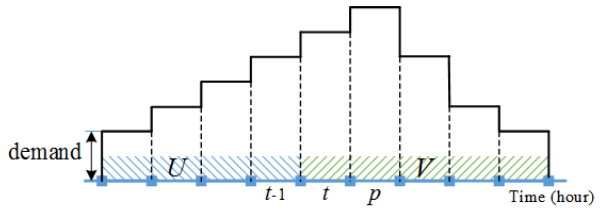
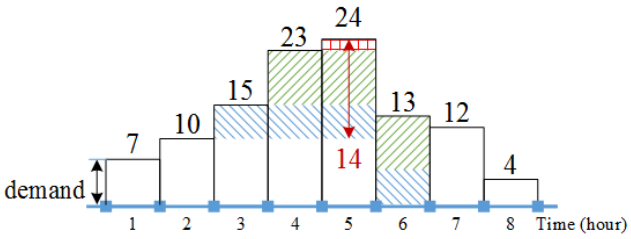


Figure 5. Basic observation.



Start Hour	Number of Servers	Minimum Active Period
1	7	hours 1 to 4
2	3	hours 2 to 5
3	5	hours 3 to 6
4	8	hours 4 to 7
5	1	hours 5 to 8

Figure 6. An infeasible demand list

if we can reserve scheduled instances to fulfil the following conditions:

1) once an instance is started, it can only be stopped after at least 4 hours.
2) the number of active instances in any hour is equal to the demand for that hour.

A feasible demand list may have multiple reservation schedules satisfying the above two conditions. We refer to these schedules as *eligible reservation schedules*. If a demand list is not feasible, no reservation schedule exists to meet all the demands without having any reservation period of scheduled instances left idle.

Our solution is based on the following observation illustrated in Figure 5. Consider a demand list with a single peak. That is, there exists an hour p such that the demand is non-decreasing before p and is non-increasing after p. If the demand list is feasible, then there must exist an eligible reservation schedule in which no instance is stopped before p and no instance is started after p. In order to prove this claim, suppose that an eligible reservation schedule for the demand list has one instance U stopped at hour $t < p$. Since the demand is non-decreasing before p, the demand at hour t is no less than that at hour $t-1$. If U is stopped at hour t (i.e., active till hour $t-1$), in order to fulfil the demand at hour t, there must be at least one new instance V started at hour t. Then, instead of stopping instance U at hour t and starting instance V at hour t, an alternative schedule is to extend the running time of instance U to cover the entire reservation period of instance V. The resulting new schedule remains eligible and has one less instance stopped before hour p. This process can be repeated to continue reducing the instances stopped before hour p until all such instances are eliminated. Thus, there exists an eligible reservation schedule with no instance stopped before hour p. Similar arguments can be made to prove that no instance is started after hour p.

A direct implication of the above observation is as follows. For each hour t, let $x_t = r_t - r_{t-1}$ be the difference between the demands of hour t and the previous hour $t-1$. If a demand list is feasible, then there must exist an eligible reservation schedule such that at any hour t, if $x_t > 0$, then exactly x_t instances are started at hour t. Otherwise, if $x_t < 0$, then exactly x_t instances are stopped at hour t. It can be shown that this result applies to any demand list and is not restricted to those with a single peak.

Recall that an instance can only be stopped after being active for a minimum period of 4 hours. Hence, a demand list is infeasible if at some hour t where $x_t < 0$, there are insufficient instances that can be stopped. This implies that at this hour t, the number of active instances would have to exceed the demand. Note that the number of instances that must be active at hour t is the sum of the positive values among x_{t-1}, x_{t-2} and x_{t-3}, i.e., the total number of instances started in the previous 3 hours. For example, consider the demand list shown in Figure 6. At hour 6, the minimum possible number of active instances is $x_3 + x_4 + x_5 = 5 + 8 + 1 = 14$, while the demand at hour 6 is $r_6 = 13$. Therefore, this demand list is infeasible. Based on this observation, we can perform feasibility testing by checking whether the minimum possible number of active instances at each hour t (where $x_t < 0$) is less than or equal to the demand at hour t.[2] If so, the demand list is feasible. If not, then the demand list is infeasible.

If a demand list is infeasible, to take the best advantage of scheduled instances, we are interested in finding an approximate solution where the scheduled instances reserved can cover the demands as much as possible. This can be achieved by removing some of the demands so that the resulting demand list is feasible. The removed demands will be covered by on-demand instances. In the following, we describe our approach.

Starting from hour $t = 1$, we iteratively compute the difference $x_t = r_t - r_{t-1}$ between the demands of two successive hours $t-1$ and t. If $x_t > 0$, then we initiate x_t instances at hour t. If $x_t < 0$, then we compute the minimum possible number of active instances at hour t. If it exceeds the demand at hour t, let d denote the exceeding amount. To make the demands feasible, we can remove d instances that should be active at hour t due to the minimum duration requirement and remove the amount of demands that these instances have covered so far (i.e., from their initiations to hour $t-1$). This will allow us to make the demands up to hour t feasible. In order to minimize the amount of demands

[2]We remark that it is actually sufficient to perform this test at hours t, $t+1$ and $t+2$ for all the hours t where $x_t < 0$ and $x_{t-1} \geq 0$. This is because the demands at hours outside the above range will not affect the feasibility result.

Scheduling Algorithm
Input: a demand list $r_1, r_2, \ldots, r_{24}$
Output: a list of tuples $[s, t]$, each tuple represents a scheduled instance to be started at hour s and stopped at hour t

```
let Q be an empty queue;
let x_1 = r_1, add x_1 copies of 1 into Q;
let t = 2;
while t ≤ 24 do
    compute x_t = r_t − r_{t−1};
    if x_t > 0 then
        add x_t copies of t into Q;
    if x_t < 0 then
        let m be the sum of positive values among x_{t−1},
        x_{t−2} and x_{t−3};
        if r_t < m then
            remove the last m − r_t items from Q and put
            them in an array T;
            for each s ∈ T do
                let r_i = r_i − 1 for i = s, s + 1, ..., t − 1;
            update x_i = r_i − r_{i−1} for i = t − 2, t − 1, t;
        remove the first x_t items from Q and put them in
        an array S;
        for each s ∈ S do
            output tuple [s, t];
    t = t + 1;
```

Figure 7. Scheduling algorithm

to remove, we order all the active instances according to their start times and remove the last d instances (i.e., those with the latest start times). Figure 7 presents the pseudo code of our algorithm to find a reservation schedule of scheduled instances for a demand list. The algorithm outputs an eligible reservation schedule if the demand list is feasible and an approximate solution if the demand list is not feasible.

For example, we have shown that the demand list in Figure 6 is infeasible. The exceeding amount at hour 6 is $14 - 13 = 1$. Thus, we can remove one active instance at hour 6. Among all the instances active at hour 6, the last started one was started at hour 5. Removing this instance and the demands it covers (i.e., one unit at hour 5) will render the resulting demand list feasible.

Finally, we remark that the idea of our scheduling solution is general and can be applied to minimum duration requirements of any length. To deal with a duration requirement of n hours, we only need to re-define the minimum possible number of active instances at an hour t as the total number of instances started in the previous $n - 1$ hours.

IV. Case Study

To demonstrate the efficacy of our cost optimization strategy, we collect the workloads of two online applications for case study.

A. Data Collection

The first application is a popular online game called League Of Legend (LOL). LOL is a multi-player battle arena video game. In this game, each session is a match consisting of 10 players. The players form two opposing teams and they fight in an arena.

We crawled the workload data of the LOL game in North America. Starting from the seed data provided by an official API [10] which includes the information of 1000 matches, we retrieved the lists of players involved in these matches, and then the lists of matches played by all these players. After that, we further retrieved the player lists of the new matches obtained and the match lists of the new players obtained. We repeated this process iteratively to get as many matches as we could. In this way, we acquired the information of more than 100 million matches played by around 1 million players. Our data covers a period of two years from 1 August 2014 to 31 July 2016. We use the number of matches played per hour as an indicator of workload. Figure 8 shows the number of matches played in each hour in four sample weeks. It can be seen that the workload exhibits a regular diurnal pattern. Daily peak loads appear in the evening. These characteristics are shared by the workloads of different weeks.

The second application is the Wikipedia Mobile (WikiM) service, which includes the mobile sites for all Wikimedia projects. We downloaded the hourly page view statistics of WikiM for two years (1 August 2014 to 31 July 2016) [11]. We use the total size of the content returned by WikiM in each hour as a workload indicator of WikiM servers.

Figure 9 shows the size of the content returned by WikiM in each hour in four sample weeks. Similar to Figure 8, the workload shown in Figure 9 also exhibits a strong diurnal pattern. Daily peak loads appear in the daytime from late morning to afternoon. The workload variations of WikiM are less significant compared to those of the LOL game. These fluctuating workloads bring a challenge for the game provider and Wikipedia operator to provision the server instances in a cost-effective manner.

B. Regularity

To evaluate the regularity of the workloads, we examine the correlations between the workloads at the same hour on different days. We use the Kullback-Leibler Divergence (KLD) [16] as a distance metric to quantitatively measure the similarity between workload distributions. Equation (1) defines the KLD between two discrete probability distributions P and Q.

$$D_{KL}(P\|Q) = \sum_i P(i) log \frac{P(i)}{Q(i)} \tag{1}$$

If the KLD is close to zero, P has a minor divergence from Q. But the KLD is not symmetric as $D_{KL}(P\|Q) \neq$

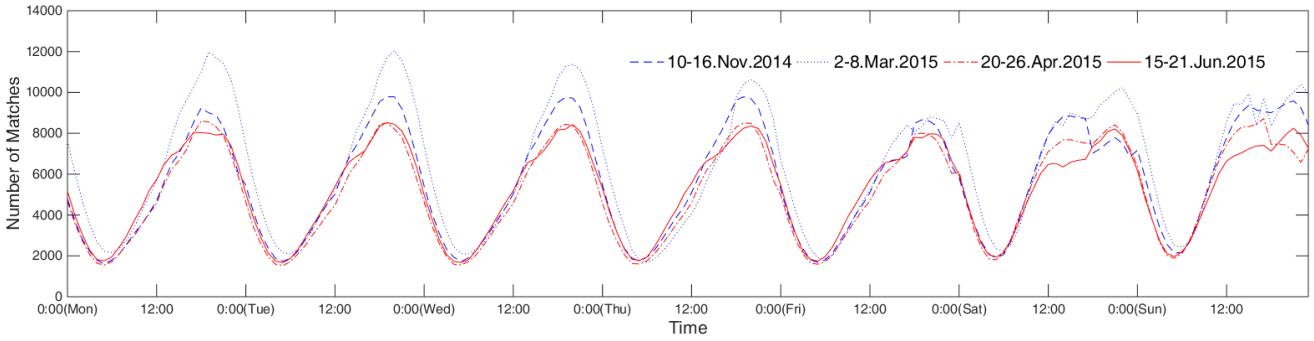


Figure 8. Sample LOL game workloads

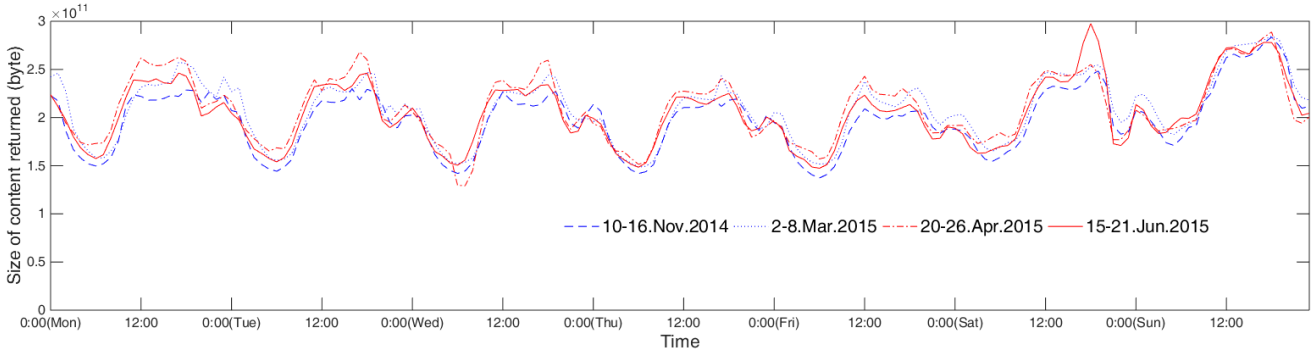


Figure 9. Sample Wikipedia Mobile workloads

$D_{KL}(Q\|P)$. Only when $D_{KL}(Q\|P)$ is also close to zero, P and Q can be considered similar to each other. To compare the workloads in the same hour on different days, we compute the workload distributions of the LOL game and the WikiM service in each hour over a week based on our workload data of two years. Since there are 24 hours per day and 7 days per week, for each application, we obtain a total of $24 \times 7 = 168$ distributions from our workload data, each with 104 samples as there are 104 weeks in two years.

Figure 10 shows four sample residual probability distributions of the LOL workload for the 9th and 22nd hours on Tuesday and Thursday. It can be seen that the workload distributions of the same hour on different days are quite similar. But the workload distributions of different hours on the same day can be very different. Table III lists the KLDs between the LOL workload distributions of the 9th hour on different days of a week. As can be seen, all the KLDs are less than 0.05. The average KLD for the 9th hour across different days is 0.02.

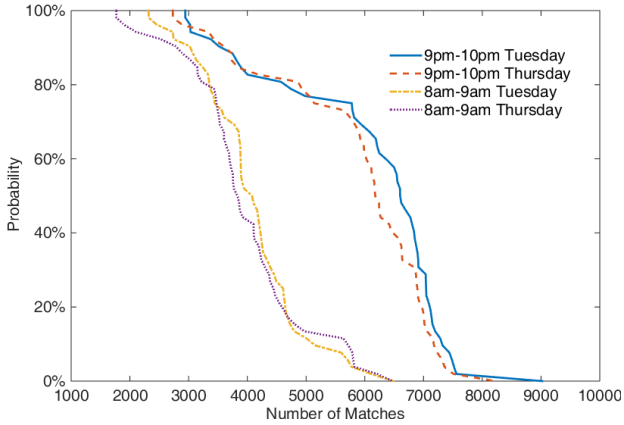


Figure 10. Residual probability distributions of the LOL workload

To illustrate the KLDs in other hours, Table IV lists the average KLD for each hour of a day for the LOL workload. As can be seen, all the values are less than 0.06. Similar trends are also observed for the WikiM workload. Table V lists the average KLD for each hour of a day for the WikiM workload. Compared with Table IV, we can see that the KLDs between the WikiM workload distributions are even smaller than the LOL workload distributions. From these

Table III
KLD BETWEEN LOL WORKLOAD DISTRIBUTIONS (8AM∼9AM)

P / Q	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Mon	-	0.0184	0.0214	0.0118	0.0273	0.0353	0.0171
Tue	0.0172	-	0.0179	0.0164	0.0381	0.0383	0.0213
Wed	0.0137	0.0163	-	0.0114	0.0233	0.0355	0.0180
Thu	0.0112	0.0181	0.0144	-	0.0210	0.0305	0.0162
Fri	0.0217	0.0342	0.0205	0.0153	-	0.0416	0.0265
Sat	0.0290	0.0341	0.0344	0.0292	0.0446	-	0.0092
Sun	0.0177	0.0235	0.0224	0.0165	0.0359	0.0135	-

Table IV
AVERAGE KLD OF LOL WORKLOAD FOR EACH HOUR

0∼1am	1∼2am	2∼3am	3∼4am	4∼5am	5∼6am
0.0250	0.0251	0.0288	0.0261	0.0199	0.0168
6∼7am	7∼8am	8∼9am	9∼10am	10∼11am	11∼12pm
0.0231	0.0251	0.0346	0.0487	0.0543	0.0354
12∼1pm	1∼2pm	2∼3pm	3∼4pm	4∼5pm	5∼6pm
0.0241	0.0229	0.0247	0.0269	0.0375	0.0428
6∼7pm	7∼8pm	8∼9pm	9∼10pm	10∼11pm	11∼0am
0.0326	0.0270	0.0202	0.0221	0.0234	0.0168

Table V
AVERAGE KLD OF WIKIM WORKLOAD FOR EACH HOUR

0∼1am	1∼2am	2∼3am	3∼4am	4∼5am	5∼6am
0.0016	0.0016	0.0018	0.0017	0.0019	0.0015
6∼7am	7∼8am	8∼9am	9∼10am	10∼11am	11∼12pm
0.0011	0.0008	0.0010	0.0009	0.0008	0.0008
12∼1pm	1∼2pm	2∼3pm	3∼4pm	4∼5pm	5∼6pm
0.0008	0.0008	0.0009	0.0009	0.0008	0.0010
6∼7pm	7∼8pm	8∼9pm	9∼10pm	10∼11pm	11∼0am
0.0012	0.0011	0.0015	0.0017	0.0015	0.0012

results, we can conclude that the workload distributions of the same hour on different days are similar to each other for the LOL game and the WikiM service. This matches the features of scheduled instances very well and justifies the need for these two applications to reserve scheduled instances with daily recurring patterns.

C. Instance Reservation for Cost Optimization

We apply our proposed cost optimization strategy to the LOL and WikiM workloads. For each application, we use the workload data of the first year as a training set to build the workload model and compute the optimal combination of the three pricing options, and then we use the workload data of the second year as an evaluation set to evaluate the reservation plan obtained.

We assume that each server instance can support 10 matches per hour for the LOL workload and can serve 0.5 GB of content per hour for the WikiM workload. Using our workload data, we compute the distribution of the number of instances needed for each hour of a day and feed the distributions into the instance allocation algorithm presented in Section III-A. Tables VI and VII show the allocation results for using the first year workload data, where R is the number of reserved instances required, and r_i is the number of scheduled instances required in hour i of a day. It is easy to see that scheduled instances are required in the afternoon and evening to cope with the higher workload of LOL, and in the morning and afternoon to deal with the higher workload of WikiM.

Then, we run the scheduling algorithm presented in Section III-B based on the demands for scheduled instances output above. Figures 11 and 12 show the scheduling results. For the LOL workload, the demand list is feasible and our algorithm finds an eligible reservation schedule to cover all the demands. For the WikiM workload, the demand list is infeasible and our algorithm finds an approximate solution by removing a little demand (3 instances) in hour 15.

Table VI
ALLOCATION RESULT BASED ON THE 1ST YEAR LOL WORKLOAD

r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}	r_{20}	r_{21}	r_{22}	r_{23}	r_{24}
27	93	128	160	182	206	254	241	231	185	113	5

$R = 469$, $r_1 \sim r_{12} = 0$

Table VII
ALLOCATION RESULT BASED ON THE 1ST YEAR WIKIM WORKLOAD

r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}
28	30	38	46	41	32	35	30	27

$R = 391$, $r_1 \sim r_8 = 0$, $r_{18} \sim r_{24} = 0$

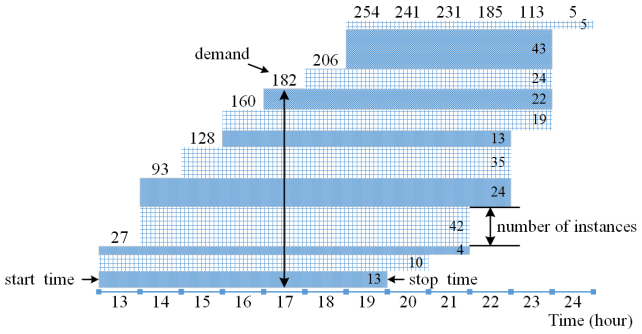


Figure 11. Scheduled instance arrangement (LOL)

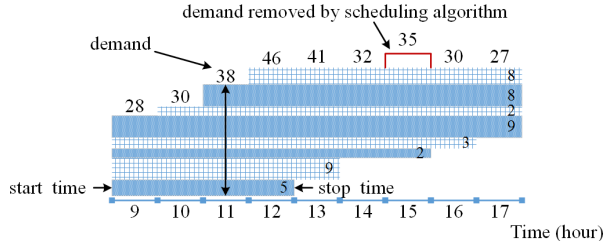


Figure 12. Scheduled instance arrangement (WikiM)

On obtaining the instance reservation plan using the first year workload, we apply it to the second year workload and compute the percentage of workload served by each pricing option as well as the total cost incurred. In addition, we also compute the optimal reservation plan directly using the second year workload and the corresponding workload division among pricing options and the total cost incurred. Figures 13 and 14 compare these results. It can be seen that the workload divisions of the two reservations plans are quite similar and their costs are pretty close. The workload divisions differ by less than 5% for both LOL and WikiM. The costs differ by only 0.2% and 0.6% for LOL and WikiM respectively. These results show that the workload distributions of these two applications are rather stable and it is practicable to use the models of historical workloads to optimize the combination of pricing options.

Finally, we evaluate the potential benefits for using scheduled instances. It is likely that the price discounts and duration requirements offered by cloud vendors for scheduled instances may change in the future. Therefore, we study the cost savings over a wide range of these parameter values.

We again use the first year workload as the training data to compute the reservation plan and use the second year workload for evaluation. Currently, the equivalent price discount of a reserved instance compared to an on-demand instance is about 43% (i.e., $1-603/(0.12\times 8760)$). Thus, we vary the price discount of a scheduled instance up to 40%. We compute the cost saving of using the best combination of the three pricing options compared to the best combination of on-demand and reserved options only. Figures 15 and 16 show the cost savings under different price discounts and duration requirements of scheduled instances. As expected, more cost can be saved with increasing discount. Note

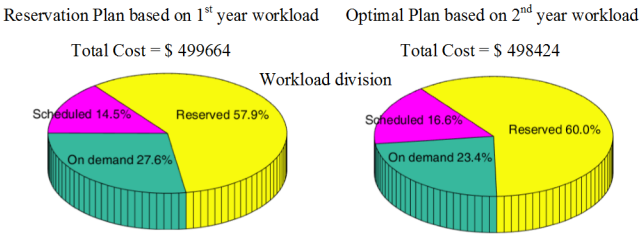


Figure 13. Workload division among pricing options and cost comparison (LOL)

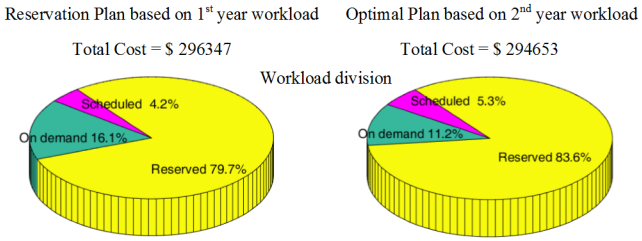


Figure 14. Workload division among pricing options and cost comparison (WikiM)

that the relation between the cost saving and the price discount is not perfectly linear. This is because the numbers of instances at different utilizations are not uniform. With an increasing price discount, a disproportionately larger number of instances can benefit from the option of scheduled instances.

In addition, Figures 15 and 16 show that the duration requirement of scheduled instances also affects the cost saving. In general, if the cloud vendor sets a longer duration requirement, less cost can be saved by using scheduled instances. The reason is easy to understand. Under a tougher duration restriction of longer time, it will be harder to find an eligible reservation schedule to fully cover the demands for scheduled instances. Then, only approximate solutions can be constructed to partially cover the demands for scheduled instances. To make a demand list feasible, the amount of demands that needs to be removed normally increases with the duration requirement. This implies that more on-demand instances may have to be used to serve the workloads, thereby reducing the cost saving.

It is also interesting to note that increasing the price discount of scheduled instances can mitigate the impact of duration requirement on cost saving. As shown in Figures 15 and 16, when the discount is below 30%, a duration requirement of 6 hours results in noticeably lower cost saving than a duration requirement of 4 hours. But when the discount exceeds 30%, these two duration requirements produce almost the same cost savings. This is because when the discount is higher, a larger number of scheduled instances are used since more workload is allocated to scheduled instances. As a result, there are more instances available for stopping to fit the demand when the demand drops across successive hours. This increases the chance for the demand list to pass the feasibility test.

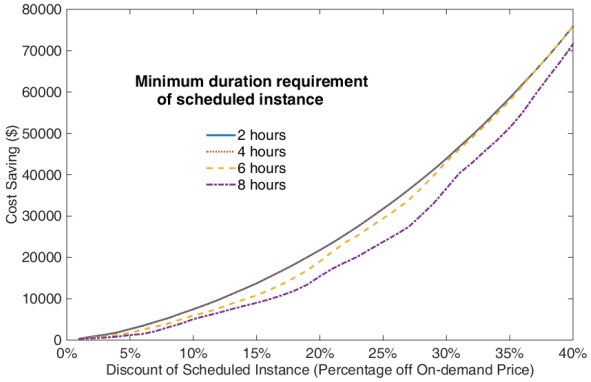


Figure 15. Cost saving of LOL workload

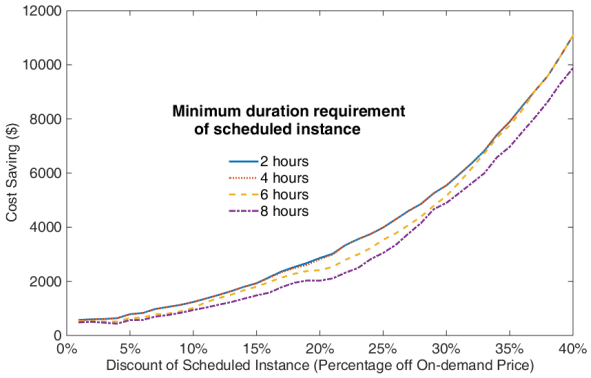


Figure 16. Cost saving of WikiM workload

V. Related Work

Cost optimization for renting cloud resources between traditional on-demand and reserved pricing options belongs to the classical rent-or-buy problems. This kind of problems involves some interesting cases like the ski rental problem [17] and the Bahncard problem [18]. Some previous work has adapted the solutions to these problems for cloud instance acquisition. To minimize the cost, Hong *et al.* [7] applied a break-even instance utilization ratio at which the cost of a reserved instance equals that of an on-demand instance to find the optimal configuration. Bodenstein *et al.* [6] introduced strategic decision models to decide at what time and with which type of pricing options to use.

To reduce the workload knowledge needed for solving the cost optimization problem, Wang *et al.* [4] proposed online algorithms to obtain a near optimal cost with limited or even no prior knowledge of the future workload. To further reduce cost, they designed a cloud brokerage service [19] to reserve a large pool of instances from the cloud provider and share it among different users. Hu *et al.* [20] proposed an online algorithm for cost optimization with multiple reservation choices that have different price discounts and contract lengths. The problem is modelled as a multi-dimensional variant of the Parking Permit Problem. However, all the reservation choices in their study are traditional reserved instances that fully book resources throughout the contracts. No scheduled instance was considered.

Besides the pricing options, another important factor for optimizing the cost of cloud users is job dispatching. As demonstrated in [21], the job dispatching strategy may significantly affect the total cost of the on-demand instances used in a cloud gaming system. The job dispatching problem for cost optimization can be modelled as a new variant of the dynamic bin packing problem [22], [23]. The First Fit packing algorithm is currently the best known algorithm for this problem [24].

VI. Conclusion

As IaaS vendors like Amazon prefer regular and steady demands, they have rolled out a new type of scheduled instances into the market. The scheduled instances make it more complicated for users to find the optimal combination of pricing options to serve a workload with minimum monetary cost. Many previous methods cannot be applied directly any more. In this paper, we have proposed a new cost optimization strategy to compute the best numbers of reserved, scheduled and on-demand instances to use given the workload distributions. Using two popular online applications as case studies, we have demonstrated the efficacy of our solution and the potential benefits of scheduled instances.

Acknowledgment

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its IDM Futures Funding Initiative, and by Singapore Ministry of Education Academic Research Fund Tier 2 under Grant MOE2013-T2-2-067.

References

[1] "Gartner says worldwide public cloud services market to grow 17 percent in 2016." [Online]. Available: http://www.gartner.com/newsroom/id/3443517

[2] "Analysts: Public cloud adoption to create major ripple effect." [Online]. Available: http://talkincloud.com/cloud-computing/analysts-public-cloud-adoption-create-major-ripple-effect

[3] "Survey analysis: How cloud adoption trends differ by geography." [Online]. Available: https://www.gartner.com/doc/3328818/survey-analysis-cloud-adoption-trends

[4] 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 10th International Conference on Autonomic Computing (ICAC)*, 2013, pp. 13–22.

[5] "Amazon elastic compute cloud - scheduled reserved instances." [Online]. Available: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-scheduled-instances.html

[6] C. Bodenstein, M. Hedwig, and D. Neumann, "Strategic decision support for smart-leasing infrastructure-as-a-service," in *Proc. 32nd Intl. Conf. on Info. Sys. (ICIS)*, 2011, p. 14.

[7] Y.-J. Hong, J. Xue, and M. Thottethodi, "Dynamic server provisioning to minimize cost in an IaaS cloud," in *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*. ACM, 2011, pp. 147–148.

[8] K. Vermeersch, "A broker for cost-efficient QoS-aware resource allocation in EC2," *Master's thesis, Universiteit Antwerpen, Antwerp, Belgium*.

[9] W. Wang, B. Liang, and B. Li, "Revenue maximization with dynamic auctions in IaaS cloud markets," in *Proceedings of the 21st IEEE/ACM International Symposium on Quality of Service (IWQoS)*. IEEE, 2013, pp. 1–6.

[10] "Riot games api." [Online]. Available: https://developer.riotgames.com/

[11] "Page view statistics for wikimedia projects." [Online]. Available: https://dumps.wikimedia.org/other/pagecounts-raw/

[12] D. Pittman and C. GauthierDickey, "A measurement study of virtual populations in massively multiplayer online games," in *Proceedings of the 6th ACM SIGCOMM Workshop on Network and System Support for Games*. ACM, 2007, pp. 25–30.

[13] S. Merritt and A. Clauset, "Social network dynamics in a massive online game: Network turnover, non-densification, and team engagement in halo reach," *arXiv preprint arXiv:1306.4363*, 2013.

[14] M. F. Arlitt and C. L. Williamson, "Internet web servers: workload characterization and performance implications," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 631–645, 1997.

[15] K. Wang, M. Lin, F. Ciucu, A. Wierman, and C. Lin, "Characterizing the impact of the workload on the value of dynamic resizing in data centers," *Performance Evaluation*, vol. 85-86, pp. 1–18, 2015.

[16] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Statist.*, vol. 22, no. 1, pp. 79–86, 1951.

[17] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki, "Competitive randomized algorithms for nonuniform problems," *Algorithmica*, vol. 11, no. 6, pp. 542–571, 1994.

[18] R. Fleischer, "On the bahncard problem," *Theoretical Computer Science*, vol. 268, no. 1, pp. 161–174, 2001.

[19] W. Wang, D. Niu, B. Liang, and B. Li, "Dynamic cloud instance acquisition via IaaS cloud brokerage," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 6, pp. 1580–1593, 2015.

[20] X. Hu, A. Ludwig, A. Richa, and S. Schmid, "Competitive strategies for online cloud resource allocation with discounts: The 2-dimensional parking permit problem," in *Proceedings of the 35th IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2015, pp. 93–102.

[21] Y. Li, X. Tang, and W. Cai, "Play request dispatching for efficient virtual machine usage in cloud gaming," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 12, pp. 2052–2063, 2015.

[22] Y. Li, X. Tang, and W. Cai, "On dynamic bin packing for resource allocation in the cloud," in *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. ACM, 2014, pp. 2–11.

[23] Y. Li, X. Tang, and W. Cai, "Dynamic bin packing for on-demand cloud resource allocation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 1, pp. 157–170, 2016.

[24] X. Tang, Y. Li, R. Ren, and W. Cai, "On first fit bin packing for online cloud server allocation," in *Proceedings of the 30th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2016, pp. 323–332.