Virtual Machines Online Acquisition

N. Alouane, J. Abouchabaka, N. Rafalia

***Abstract*—Clouds basically offer a set of instance acquisition solutions, it's either an on-demand plan where the user has to pay the full VM hourly pricing or can go with a commitment for a X duration, then the user can benefit from a Y percent of reduction over the total VM reservation period. That point of shifting or decision making becomes more difficult during the last couple years, with this big number of service reservation offers with various durations that we have on the market today and knowing the fact that not all workloads are easy to predict, it forces the user to think about an optimal combination of these offers, while maintaining the same availability level, consistency and latency of the on-demand solution. In this paper, we introduce two deterministic algorithms for the multi-slope case, that incur no more than $1+\frac{1}{1-\alpha}$ and $\frac{2}{1-\alpha}$ respectively, compared to the cost obtained from an optimal offline algorithm, where α is the maximum saving ratio of a reserved instance offer over on-demand plan. Our simulation driven by the google cluster usage data-trace shows that more than 30% of cost savings can be achieved when applied to a real cloud provider like amazon web services, while 40% when purchasing instances through a cloud broker service.**

Index Terms; online programming; cloud ec2 reservation; optimization; virtual machines

I. Introduction

The number of companies that relies on cloud services is on a rapid growth path. According to [10], By 2016 over 80% of enterprises globally will be using Infrastructure as a service (IaaS), also the Gartner's 2015 CIO survey estimates that 83% of non-using cloud services companies consider cloud IaaS as an infrastructure option, and this is because the cloud IaaS becomes more suitable for almost all hosting use cases either for development, testing environment, high-performance computing, batch processing or mission-critical business applications, and it can be used to run most workloads. However, IaaS cost management still a headache for cloud users, they find a lot of difficulties for choosing the most cheaper and suitable cloud provider for their needs (e.g., Amazon services Ec2, Rackspace Hosting, Microsoft Azure, Google App Engine…) but in reality, the cloud provider choice is not so important, since the difference between cloud providers offers is so tight if not equal. In fact the pricing models with different commitment levels along with the instance types scheduling should be the primary concerns for users, for example if there is an application that needs 7 vCPU and the three instance types provided by the cloud provider could respectively complete (Large, 4 vCPU, 3.1$), (Medium, 2 vCPU, 2.5$) and (Small, 1 vCPU, 1.2$), it would be more profitable (even if it is over-provisioned) if this user picks up two large instances instead of choosing one instance from each type. Also, where a heavy cloud user can rely on reserved instances marketplace where he can reserve an instance for months while taking advantage of significant reductions (up to 60%), instead of using *on-demand* instances and pay only for the incurred instance-hours. So, we can see clearly that the virtual machines (VMs) purchasing strategies is very important either for a heavy cloud user or a cloud broker in order to take full advantage of cloud services. In this paper we focus on VMs purchasing strategies based on different pricing models with different commitment levels, and we answer tree important questions: what type of commitment level should I reserve, 2- when should I reserve 3- and how many VMs should I reserve?

In the literature, instances of the VMs purchasing problem has been solved in most cases by either using exact historic workloads as a reference or relying on long-term prediction of future workload, but unfortunately even if we disregard the fact that workload is very unpredictable and unstable, a practical implementation of these solutions requires a very long prediction over time (say years), which is not always easy to get especially for start-up companies (i.e., if a user wants to make a decision about choosing a three years of commitment, it would require at least three years of workload history).

Recently, [6] proposed for the first time an online algorithm inspired from the *Bahncard problem*[13] for reserving instances with no a priori knowledge of future workload, but unfortunately the reservation strategy used in this approach is deprecated by Amazon Ec2 and no longer valid, the old Amazon Ec2 business model for reserved instances offers three utilization sizes: *1-Light utilization*: It offers the lowest upfront payment in return of receiving a significant discounted hourly usage fee, also the reserved instance can be turned off at any point without paying the hourly fee, *2-Medium utilization:* the user pays a higher upfront than light utilization in return of a much lower hourly usage, here again the user can shut down the reserved instance at any time without accumulating any fees, *Heavy utilization:* refers to the most profitable offer and most appropriate for stable workloads for a long period, the user pays a higher upfront but in exchange he benefits from the lowest hourly usage, however the user will be charged for every hour of the reservation period even if the instance is turned off. Recently Amazon discovered [11] that more than 95% of customers are choosing the third model "*Heavy utilization*", thus, they have changed the business pricing model, and right now users are given the choice between: paying the entire reservation period as an upfront, paying half of the reservation period as an upfront fee while the remainder is split over the following months, or paying no upfront but the entire reservation cost is split over the following months, and of course the user is still charged over all the reservation period either the instance was turned on or off. So, this changing over the business model of reserved instances has changed things, especially, the competitive

Manuscript received Sept 26, 2017; revised Oct 10, 2017.

N. Alouane is PhD student in Laboratory of computer science and telecommunication in Ibn Tofail University, Morocco (corresponding author phone: +212640454375; e-mail: alouane00@gmail.com)

J. Abouchabaka is Full Professor of Computer science at the University of Ibn Tofail, Morocco (email: abouchabaka3@yahoo.fr).

N. Rafalia. is Full Professor of Computer science at the University of Ibn Tofail, Morocco (email: arafalia@yahoo.com).

ratio of online algorithms designed to solve the VMs purchasing problem, the gap between the online and the optimal instance acquisition algorithm will certainly increase. Another problem that we can mention is related to the reserved instances location, all previous works assume that either a cloud broker or a user allocates instances within the same region or the same availability zone (AZ), but in reality, it is not always true. The Amazon Ec2 policy allows switching of reserved instances AZs only within the same region, so if we take this fact in consideration, users that require instances hosted in different regions due to some latency problems would not fully benefit from the RI discount (e.g., in the worst case, the RI would be located inside an inactive region, and the user will be charged even if the RI is turned off).

In this paper, we extend the work of [6] by taking in consideration the new business pricing model used by Amazon EC2, and we solve the problem of RIs scheduling with different commitments level for the Multi-Slope case. To our best knowledge this is the first work that address the problem of RIs scheduling in an online manner while considering multiple reservation offers. So, in summary we make the following contributions:

- We prove that the RIs scheduling problem is indeed NP-hard, by using a reduction from the *longest path problem* [12]
- We prove that the competitive ratio of any deterministic online algorithm is at most 2 times the minimum cost obtained by an optimal offline algorithm that knows the exact future a priori
- We propose two deterministic algorithms that incurs no more than $1+\frac{1}{1-\alpha}$ and $\frac{2}{1-\alpha}$ respectively where α is the maximum saving of a reserved instance offer over on-demand plan

II. Related works

In the literature, many approaches and techniques have been designed in order to reduce the user's IT computing cost, some of them, like [10] are focused on instance types scheduling, their objective is to find the optimal combination between VMs types (i.e., *Large*, *Medium* or *Small*) to fill the user's capacity request within a time far less than brute force method (i.e., testing all types of VMs combinations). However, brute force method is still an effective solution, even the smallest instance from Amazon Ec2 (i.e., t1.micro) can run the brute force scheduling algorithm for more than a thousand type of VMs in less than an hour, which is the minimum subscription time, so we concluded that VMs type scheduling is not crucial for users. In this section, we focus rather on works that provide a rental planning between on-demand and reserved instances plan to reduce the instance acquisition cost. We also give insights about previous works around the multi-slope rental problem.

A. VMs purchasing strategies

In [2] the authors addressed the problem on how a web application should plan the long-term reservation contracts in such a way that the user profitability is increased. Different tests were conducted in case of high, normal and low workload of a web application, however their model is completely depended on administrator's inputs like max, min, and average of workload, also the downtime penalty estimation which is in practice may not be available, and very hard to compute especially for start-up companies.

[3] Assumes that the future workload is known in advance, and propose two possible remedies to the problem of VMs scheduling based on the type and the subscription time, the first solution was to simplify the problem by fixing the minimum subscription time of all instance types to an equal length, thus the problem was reduced to an integer programming problem and can be solved in a matter of seconds even for real problems. In the second solution, they propose a heuristic solution of the problem with heterogeneous subscription times (e.g., 1 hour, 1 day, 1 week, 1 month…). But here again as we mentioned before the workload prediction is very hard to get and not reliable.

In [7], authors proved that finding the optimal VM renting strategy along with the jobs scheduling problem are computation intractable and introduce a new approximation algorithm for minimizing the computing cost for deadline-constrained batch jobs. But their approach assumes that the workload is known in advance. Moreover, in their experimental results the deadline time was bounded between one month and two-month whereas in reality the deadline time is much less than that.

The first and only work that addressed the problem of instance renting strategies in an online manner without a priori knowledge of the future workload was [6], they first proved that even the optimal strategy where the entire future demands are given, suffers from the "curse of dimentiality" and is computationally intractable, and they left open to show whether the offline problem is NP-hard. They proposed a deterministic (resp., randomized) algorithm that incurs no more than $2-\alpha$ (resp., $\frac{e}{e-1-\alpha}$ times the cost of the optimal offline algorithm. However, their approach suffers from several limitations: 1)- the business pricing model adopted in their work is no longer used by amazon EC2, 2)- they discussed the case of one single renting option, which reduces the complexity of the problem.

B. Muti-Slope Rental Problem

In this section, we briefly review some research efforts around the multi-Slope rental problem, and the competitive ratio reached by each approach. This review inspired us in finding a new online strategy and applying it in the cloud computing area.

Azgar et al. [1] addressed the multi-slope rental problem for the convex case, their purpose was to reduce the cost of engines provisioning in a factory, and they assumed that slopes (i.e., engines) become available over time, and the transition cost between states is the same. The obtained online algorithm has guaranteed around 6.83 as a competitive ratio. Bejerano et al. [4] considered the problem of routing ATM networks inside virtual channels (VCs), they give a 4-determinisitic algorithm for the convex and non-additive case of the multi-slope problem. Damaschke et al. [5] treated the non-additive case of the problem from [1], where moving to another slope involves new fees, they defined an upper bound of 4 and a lower bound of 3.618 for deterministic algorithms, also a randomized algorithm was presented that guarantees a 2.88 as a competitive ratio. However, none of the three works [1], [4] and [5] took in consideration the slope duration as a parameter, they all assume that the acquisition of a slope is absolute and not time-restricted. Meyerson [8] has considered slopes duration as a parameter for the parking permit problem, each permit allows a usage of some duration with different renting price,

Meyerson shows that no deterministic algorithm can do better than Θ(k) where k is the number of permits (i.e., linear in the number of permits), and no randomized algorithm can do better than $\Theta(\log k)$. Another example where the slopes duration has been taken into account is Guiqing et al. [9], they addressed the multi-ski rental problem with multiple discount options, each with a rental duration, they showed that there is no deterministic algorithm can have a small competitive ratio lower than 4 when the number of slots is large.

III. Preliminary

In this section, we introduce some details about concepts used in this paper. We firstly describe the new business pricing models used by Amazon Ec2 service. Then we explain how the competitive ratio between an online and optimal algorithm can be measured, and finally we introduce our main problem that is the multi-slope rental problem for an optimal cost management.

A. Ec2 Business pricing models

Amazon Ec2 provides customers the ability to choose between three different purchasing models for a better flexibility to optimize cost.

On-Demand plan: allow users to pay a fixed rate for compute capacity without commitment or any upfront fees. However, it is possible that users will not be able to launch a large number of on-demand instances for a short period when congestion arises in some Availability zones. On-demand instances are recommended for applications with short, unpredictable and spiky workload, also for a testing environment.

Reserved instances plan: compared to on-demand instances, users can benefit from significant discounts up to 60% for a long period (for one or three years), while the availability is guaranteed at 100% inside the chosen availability zone, users can move the reserved instance between AZs within the same region, changes its network configuration, or even sell it in case the user does not need it anymore, in the reserved instances marketplace. However even if this user turns off his reserved instance for some duration, he will be charged over all the reservation period either the instance was turned on or off. Reserved instances are more profitable for applications with a stable and steady workload.

Reserved instances marketplace: users can sell their reserved instances on behalf other users if they have been active for at least 30 days, and at least one month is remaining in the term of reservation. Unlike the RIs sold by amazon where the commitment duration is set to either 1-year or 3-years, RIs sold by users can be found in different commitment levels, between 1 month and 3 years.

Spot instances: like on-demand instances, they can be used without any upfront commitment but at a very low hourly rate (up to 80% of discount) but with the risk of sudden machine shut-down. In this paper, the spot business model is not taken into consideration due to its complexity, and very hard to anticipate.

B. Competetive analysis

Competitive analysis foundation lies on the comparison between an online algorithm and an offline optimal algorithm. An online algorithm tries to solve a given problem without knowledge of future requests or future input sequence, while an offline algorithm, act while assuming that the exact future demand is known a priori. An online algorithm alg is *c-competitive* if the performance produced by alg on any input sequence is at most c times what an offline optimal algorithm can do while using the same input. Of course, the offline algorithm is always performing better than any online algorithm, and this is why the competitive ratio is always higher than 1. Computing the performance of either an online or offline algorithm cloud be an easy exercise as it can be a tough task, especially, when it comes to *NP-hard* problems, where even finding an algorithm that can solve the offline problem in a polynomial time is impossible. So in this case the most suitable solution is working with boundaries, by looking for an upper bound and lower bound respectively for the online and the offline algorithm.

In our case a formal definition would be:

$$Cost_{alg}(R) \leq c.Cost_{opt}(R)$$

Where $Cost_{alg}(R)$is the accumulated cost by choosing algorithm alg for solving the RIs scheduling problem for any input sequence $R = \{r_1, \ldots, r_n\}$, and $Cost_{opt}(R)$ is obtained by solving the problem in an offline manner.

IV. Reservation scheduling problem (problem statement)

The reservation scheduling problem is a variation of the traditional Multi-slop rental problem, where there are multiple renting options, each option is characterized by four parameters:

b_i: Buying cost, r_i: rental cost, d_i: option duration and t_i: time arriving

There are a lot of known variations of the main problem like the parking permit problem [8], Rent-or-By problem, Multi-slop ski rental problem [9], On capital investment [1], also the Bahncard problem and the ski rental problem as a special case when there are only two options. Each problem is subject to restrictions like:

- *Additive*: moving from option i to option j requires paying the difference in buying prices $b_i - b_j$
- *Non-additive case*: transition between option i and option j is subject to a defined transition cost b_{ij}
- *Convex*: for $i < j$, $b_i < b_j$ holds when $r_i > r_j$
- *Non-convex*: the convex restriction do not necessarily holds.

Also, the option's time arriving parameter is rarely considered in most problems, due to its complexity and it only increases the competitive ratio between the offline and online algorithm. In this paper, we omit the time arriving parameter and assume that all reservations options are available at time 0.

The reservation scheduling problem can be formulated as the following: We have n different reservations contracts that can be purchased, each contract can be represented by $R < b_i(\$), r_i(\$), d_i(months\ or\ hours) >$. So, if we take the $c1.medieum$ instance from Amazon Ec2 as an example, an on-demand offer would be$R < 0\$, 0.5\$, \infty >$, whereas a reservation contract of 3 months would be$R < 80\$, 0.2\$, 3months >$. It is obvious that using on-demand offer is more efficient for high fluctuating and sporadic workload, while long term reservation contract is more suitable for stable requests lasting for a long period. However, in cases where history and reliable predictions are

unavailable (like start-up companies) making purchase decisions with caution and in an online manner is very important. So, our goal is to optimally combine between different reservations contracts to serve every type of requests while minimizing the competitive ratio c of the cost incurred by our algorithm versus the cost incurred by the offline algorithm which see the future requests in advance.

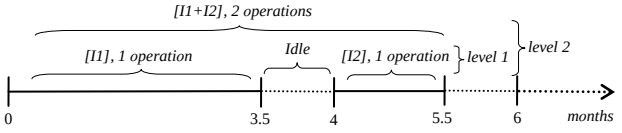


Figure 1. Example of a user requesting a single VM during two separate intervals I1 and I2.

A. *Offline reservation problem analysis*

Solving the offline reservation problem requires solving two sub problems: rental planning problem for one VM at a time and optimal strategy for distributing idle reservations instances for other requests.

1) ***Rental planing problem for one VM at a time:***

In this section, we prove that the first sub problem is not NP, thus it can be solved in a polynomial time. Figure 1 illustrate an example where the user asks for a single VM instance at a time, for example in the first interval where the demand duration is 3.7 months, the offline algorithm has two choices (we notice that the problem is convex): using a 3 months of reservation commitment and using on-demand offer for the remaining period (i.e., 0.7 month), or using a 4 months of reservation commitment (even if it is overprovisioned). We count this comparison between the two solutions as a single operation. Also in the second interval (i.e., 1.2 months) we have two choices either renting one instance during a whole month while using on-demand offer for the remaining period (i.e., 0.2 month) or renting an instance for two months of commitment, and here again we have one operation to curry. However, if we consider the interval *I1 + I2* we have three options: using the best option of *I1* (computed previously) along with the best solution of *I2* separately, renting an instance for 5 months of commitment while using on-demand offer for the remaining 0.4 month, or renting an instance for 6 months of commitment. So in this level we have two operations to execute, in the same way, we can analyse the remaining intervals and conduct the optimal rent strategy. Computing the complexity of this algorithm is an easy exercise, we have just to compute the number of operations in each level (m is the number of intervals):

level 1 => m ops, *level2* => 2(m-1) ops, *level3* => 3(m -2) ops … *level* m => $m\big(m-(m-1)\big) = n$ ops

So, the complexity of this algorithm $= m + 2(m-1) + \cdots + m\big(m-(m-1)\big) \sim n^3$

This serve as an evidence that the rental planning problem for one VM at a time can be solved in a polynomial time.

2) ***Idle reservation instances distribution (IRID)***

In this section, we prove that distributing idle reservation instances problem is actually NP hard, and thus there is no algorithm that can solve the problem in a polynomial time (assuming **P ≠ NP**). In Figure 2 (a) we split the instances requests into a set of multiple levels (i.e., *level 1*, *level 2*...), each level asks for a single VM at a time (e.g., *level 4* requests one single VM during the interval [5, 6.4]), afterwards we optimally solve the rental planning problem by using the previous algorithm Figure 2 (b) (e.g., in *level 4* it is more profitable using two months of reserved instance instead of one, while the remainder (*0.4 month*) is filled with on-demand instance).

However, we see that there are some idle reservations remaining in each level, and we should specify how those idle reservations should be distributed towards other levels. For example, in Fig 2 (c) it can be more profitable using the idle reservation obtained from *level 2* to fulfil the *level 1*'s demand, in this case we would have another planning partition for *level 1*, that is, finding the optimal renting planning for the two intervals [5, 8] and [10, 12.5] separately, as a result we get a 3 months RI for interval [5, 8] and a 5 months RI for interval [10, 12.5], and of course if we get other idle reservations from this new planning we have to repeat this operation until we reach the bottom of the diagram (*we note that level 3's idle reservation -figure 2(b)- can be also be used to fulfil other levels like (level2, level1, …).* So, the main question right now is, starting from Figure 2(b), what is the optimal idle RIs distribution?

In the following we prove that this problem is NP hard (but not NP-complete, since it isn't a decision problem) using a reduction from the *longest path problem* [X], which is a known NP hard problem.

Definition: Given a weighted and oriented graph G without negative cycles, a path between two vertices (u, v) is called *simple* if it does not have any repeated vertices. The longest path problem refers to the longest simple path between two vertices (u, v).

In the following we describe the reduction from IRID problem into a graph that has the longest path of a certain size if and only if IRID has an optimal solution:

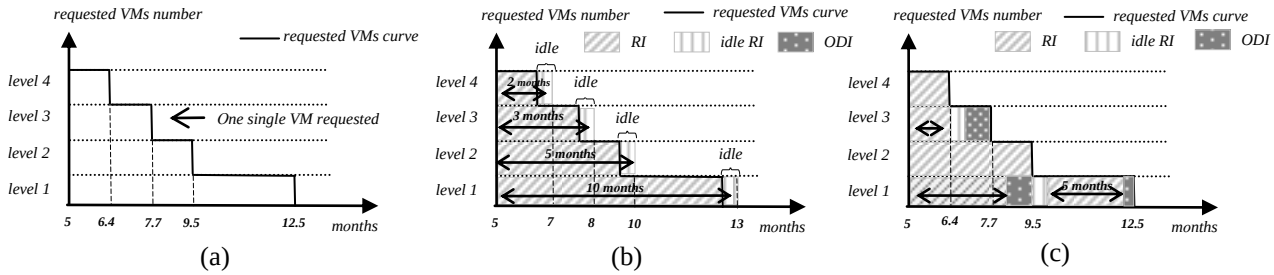


Figure 2. Example of an offline algorithm behavier for reserving instan

1. Each node represents a spot that can be filled with an
2. idle reserved instance, it can also represent the case where an idle RI is not used at all.
3. Two nodes (n_1, n_2) are connected by an oriented edge $(n_1 => n_2)$ if n_1's idle RI comes just before n_2's idle RI, and they are not related to the same idle reserved instance (i.e., using n_1's idle RI will not affect the existence of n_2's idle RI). An edge can be attached to a restriction (see Figure 3 (b), edge (*nB2*, *nC1*)
4. The weight of an oriented edge between two vertices (n_1, n_2) is the gain accumulated when using the idle reserved instance in spot n_2, compared to not using the n2's idle RI at all
5. Add a (start, end) vertices to graph *G*

In Fig 3 we illustrate the reduction from the previous problem (Figure 2 (b)) into a graph, first we start by creating a new node in each spot that can be used to run an idle RI, see Figure 3 (a), besides, we add nodes like *nA*, *nB* and *nC* to manage cases where idle RIs: *A*, *B*, and *C* are not used, see figure 3 (b).

We can see that *nA1*, *nA2* and *nA3* are not connected since we can use the idle reserved instance *A* to fulfil only one of those nodes, besides *nA1* is not connected to nodes like *nB1* and *nB2* because using the idle reserved instance *A* for *nA1* spot incurs changes in planning partition of *level 2*, which means it is possible that there will be no idle reservation instance *B* and no *nB1* and *nB2* possible spots. Finally, we add a *start* node and *end* node to graph G, and we try to find the largest path between those two nodes, which correspond to strategy with the optimal and largest reduction possible.

The reduction from the largest path problem to IRID is trivial (i.e., solving Figure 3 (b) problem in a polynomial time is equivalent to solving the *longest path problem*)

I. Deterministic Online Algorithms

In this section, we present three online deterministic algorithms, we obtain from the first algorithm a lower bound where one cannot expect more than $\frac{1}{1-\alpha}$ times the optimal cost, where α represent the maximum saving of a reserved instance contract over on-demand plan, and it follows two interesting algorithms with different reservation strategies, we proceed to talk about either risk taking or proceeding with caution.

A. Rent-Never-Algorithm(RNA)

This algorithm suggests that we never rent an instance and use only on-demand instances.

Theorem: This algorithm is $\frac{1}{1-\alpha}$ competitive.

Proof: Consider $\{sav_i\}_{0\le i\le n}$ as the saving of a reserved instance $R < b_i, r_i, d_i >$ over on-demand plan, we note α as the largest saving over all RIs. In the worst case, each *level* will need an instance for a long duration, more than the longest reservation period (e.g., three years in case of Amazon Ec2), so the competitive ratio will be:

$$c = \frac{Cost_{rna}(t)}{Cost_{opt}(t)} = \frac{Cost_{rna}(t)}{(1-\alpha) * Cost_{rna}(t)} = \frac{1}{1-\alpha}$$

As a direct application in Amazon EC2, α equals 57%, and the competitive ratio of *RNA* algorithm is ~ 2.32

B. Lower bound for deterministic algorithms

We believe that RNA's competitive ratio is also a lower bound for any deterministic algorithm, by unfortunately we couldn't find an appropriate formal proof for that. However, we give a lower bound value of 2, just like other works with a similar multi-slop problem.

We first discuss the problem with two slopes and then give our generalization to the multi-slop case:

Let $R_0 < b_0, r_0, \infty >$ and $R_1 < b_1, r_1, d_1 >$ be the only two available offers, and x is time of transition from R_0 to R_1, any deterministic online algorithm has three choices:

1. Rent R_1 before $\frac{b_1+r_1*d_1}{r_0}$ instance requests (this is the break-even-point between R_0 and R_1, we will talk about it later)
2. Rent exactly when the instance requests number equals $\frac{b_1+r_1*d_1}{r_0}$
3. Rent after $\frac{b_1+r_1*d_1}{r_0}$ instance requests

In case 1: $\frac{Cost_{case1}(t)}{Cost_{opt}(t)} = \frac{r_0*x+(b_1+r_1*d_1)}{r_0*x} \le 2$ since $x \le \frac{b_1+r_1*d_1}{r_0}$

In case 2: $\frac{Cost_{case2}(t)}{Cost_{opt}(t)} = \frac{r_0*x+(b_1+r_1*d_1)}{r_0*x} = 2$ since $x = \frac{b_1+r_1*d_1}{r_0}$

In case 3: $\frac{Cost_{case3}(t)}{Cost_{opt}(t)} = \frac{r_0*x+(b_1+r_1*d_1)}{b_1+r_1*d_1} \le 2$ since $x \ge \frac{b_1+r_1*d_1}{r_0}$

Let us now consider the multi-slope case. In case all the reservation offers are very close except the last one:

$$(b_i + r_i * d_i) \sim r_0 * d_i \quad s.t.\ 0 \le i < z \qquad (1)$$

The lower bound of any deterministic online algorithm (*doa*) will be:

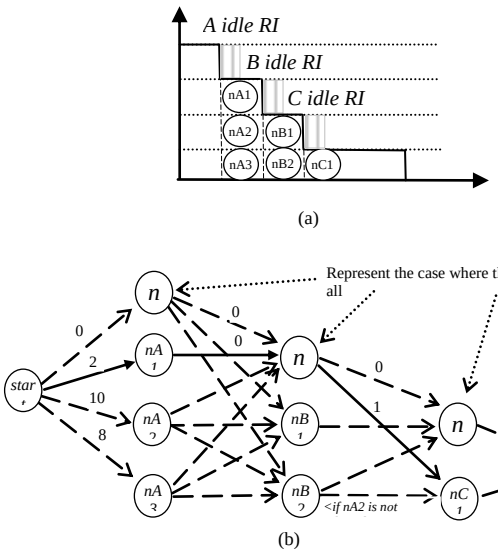


Figure 3. Reduction from the IRID problem to the longest path problem, in (b) the thick line indicates the optimal cost as a function of the game duration time

$\frac{Cost_{alg}(t)}{Cost_{opt}(t)} = \frac{(b_i+r_i*d_i)+\cdots+(b_p+r_p*d_p)}{Cost_{opt}(t)}\ s.t. 0 \le i < p$

Where p is the number of all reservation offers used by a *doa.*

If alg wants to make a good competitive ratio it has to use $R_z$offer, and by considering(1), we come back to the previous problem with only two reservation offers:

$$\frac{Cost_{alg}(t)}{Cost_{opt}(t)} = \frac{r_0 * (d_i+..\ d_{p-1}) + (b_z + r_z * d_z)}{Cost_{opt}(t)}$$

We believe that the resulted lower bound is not optimal, and it can be more than 2.

C. Maximum-Duration-Algorithm(mda)

This algorithm uses a reservation option $R_i < b_i, r_i, d_i >$ if and only if the last longest and uncut period of instance request is greater than or equal to di. Initially all instances requested by all *levels* are supposed to use on-demand plan, and let:

D_t: the number of instances requested by the all *levels* at time t

$< R_i, rem_i >$: a RI in use, where rem_i is the remaining time of the reservation period

$Check(x)$: a check function, it equals 0 if x is false and 1 otherwise

$ReservationSet(t)$*:* is the set of reserved instances in use at time t

$ResN_t$: is the length of the $ReservationSet(t)$ set at time t

1. Initialization $t = 0$, $ReservationSet = \{\}$
2. If i existe such as $i = argmax(d_i |\ d_i \le \sum_{t-d_i-1}^{t} Check(D_t > ResN_t))$ then
3. Reserve $R_i$offer, and add $R_i$to $ReservationSet$
4. For each $< R_i, rem_i >$ in $ReservationSet$ decrement di value and remove R_i from $ReservationSet$ when $rem_i = 0$
5. Lunch on-demand plan for the remaining instances
6. Set $t = t + 1$, go to *step 2*

In this algorithm, reservation is lazy, the algorithm does not reserve until it sees that the optimal offline algorithm has already spent more or equal to what mda is about to rent. Fig. 4 helps to illustrate algorithm *mda:* if a certain *level i* asks for an instance lasting exactly eight months, obviously the optimal offline algorithm will pick a RI from the marketplace that last exactly eight months, however, mda will start by using on-demand instance until it reach the first RI duration in the marketplace, and rent it, which is in this example equals to one month, afterwards, it rent an instance for two months by the same way, and since mda is an online algorithm and has no a priori knowledge of future workload, it will rent a new RI for four months by assuming that the workload is still going.

In Figure 5 we illustrate the performance of mda over the previous example (Figure 2 (a)), and we see that we have more *idle* periods, and specially in *level 2*, the four months reservation is done before that *level 2* stop requesting VM, therefore mda was assuming that the VM requesting will

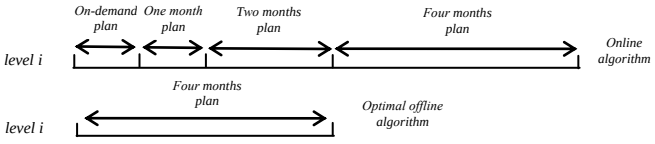


Figure 4. Illustration of algorithm 1 for a single *level i*

Theorem: This algorithm (mda) is $1 + \frac{1}{1-\alpha}$ competitive

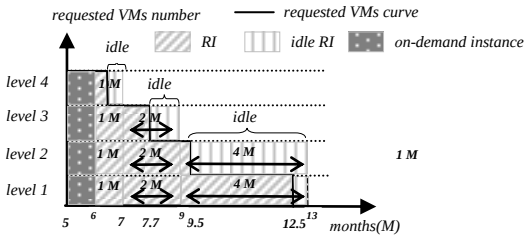


Figure 5. Applying mda to the previous example (Figure 2 (a))

continue. We note that designing an *idle* RIs distribution strategy is not possible for any online algorithm, because it is impossible to know either if *level i* will request an instance during its *idle* period or not, so in summary the online algorithm cannot even consider this period as *idle.*

Proof: In order to compute the competitiveness of this algorithm we split $Cost_{mda}(t)$ in two parts: $Cost_{mda}(t) = Cost_{mda}(t^-) + Cost(rem)$

Where $Cost_{mda}(t^-)$ denote the cost accumulated by mda until time t^- (t not included), and $Cost(rem)$ is the remaining fee of the last RI. We can write:

$$\frac{Cost_{mda}(t^-)}{Cost_{opt}(\mathrm{t})} \le \frac{Cost_{mda}(t^-)}{(1-\alpha)Cost_{mda}(t^-)} \le \frac{1}{1-\alpha}$$

In the worst case, *level i* will stop requesting instance in time t just after that mda rent a new RI, let's called $R_m < b_m, r_m, d_m >$.

so: $Cost(rem) = Cost(R_m) = b_m + r_m * d_m$

Now we have to compute the $Cost(rem)$ competitive ratio. At a certain time, let $d_{user}(t) = \sum_{t-d_i-1}^{t} Check(D_t > ResN_t)$. From step 2, we can see that d_i which is the period of the next RI is always lower than $d_{user}(t)$, so we can write: $d_m \le d_{user}(t)$, and since the optimal offline algorithm has to pick a cheap reservation offer, let's called $R_k < b_k, r_k, d_k >$ where $d_k \ge d_m$, we can write: $Cost(R_k) \ge Cost(R_m)$ because the problem is convex, therefore $Cost_{opt} \ge Cost(R_h) + \cdots + Cost(R_k) \ge Cost(R_k) \ge Cost(R_m)$

and we conclude: $Cost_{mda}(\mathrm{t}) \le (1 + \frac{1}{1-\alpha}) Cost_{opt}(\mathrm{t})$

D. Break-even-point -Algorithm(bepa)

In this algorithm, a transition between two reservations offers $R_b < b_b, r_b, d_b >$ and $R_a < b_a, r_a, d_a >$, where $a < b$ depends on a *break-even point* defined by:

$$Bep_{b,a}^{t} = \frac{b_b - b_a - \sum_{i=0}^{q} Cost(R'_i) + r_a * \sum_{i=0}^{q} d'_i}{r_a - r_b}$$

Where $\{R'_t\}_{0\le i\le q} < b'_i, r'_i, d'_i >$ is the set of RIs used previously form $t - d_b$ to t, and $Cost(R'_i) = \left(r'_i + \frac{b'_i}{d'_i}\right) * (d'_i - idle_i)$, and $idle_i$ is the *idle* period during R'i

Computing $Bep^t_{b,a}$ is simple, it represents the time where the cost accumulated when using R_a and R_b are intercepted. In Fig.6 we have an example of how to compute $Bep^t_{b,a}$, each reservation plan is represented by a line which indicates the cost incurred if an online algorithm stays in that reservation plan. So $Bep^t_{b,a}$ is the time of transition from state a to state b, and formally it is the solution of the equation:

$b_a + r_a * (x - previous\ d'_i) + Cost(previous\ R'_i) = b_b + r_b * x$

The idea behind this algorithm is simple, once R_a usage is over, we ask this question: can we get a cost saving if we had used R_b offer for the last d_b period? If the answer is yes, we should fixe our "mistake" and rent R_b if a certain condition is verified, we will talk about it later, if the answer is no, then we have to stick with the current offer, but again if a certain condition is verified. Initially all instances requested by all *levels* are supposed to use on-demand plan, and let:

c: Index of current RI

n: Total number of RIs available

//D(c,user): number of times an instance has been requested during the current RI

D^t_i: Number of times an instance has been requested during the interval $[t - d_i, t]$

$idle^t_c$: *idle* period during the current RI

A formal definition of the algorithm would be:

1. Initialization $t = 0$
2. For each *level i*
3. If R_c is over, or *level i* is on the on-demand plan
4. For i from n to c *//for each RI offer*
5. **if**$(D^t_i \ge Bep^t_{i,c}$ and $Bep^t_{i,c} < d_i)$
6. $ND = D^t_c - (d_i - Bep^t_{i,c} - d_c)$ *//new demand*
7. **if**$(ND \ge idle^t_c)$ *//idle demand*
8. Reserve R_m offer
9. **else**
10. $ADC = ND - r_c * idle^t_c$ *// Additional cost*
11. $Budget = Cost(R_i) - ADC$
12. Find R_j where $Cost(R_j) < Budget$ and $j < i$
13. **end else**
14. **end if**
15. **else**
16. $Budget = \sum_{i=0}^{q} Cost(R'_i) + r_c * (D^t_c - idle^t_c)$
17. **if**$(Budget > Cost(R_c))$
18. Reserve R_c *//Re-Reserve R_c*
19. **else**
20. Find R_j where $Cost(R_j)$ *//budget cost*
21. **end else**
22. $t = t + 1$
23. go to *step2*

Figure 6, helps to illustrate bepa. It starts with an on-demand plan, and whenever it sees that its cost exceeds $Cost(R_1)$, then such use of R_0 is not justified, and we should have reserved R_1 beforehand at time 0 and used to

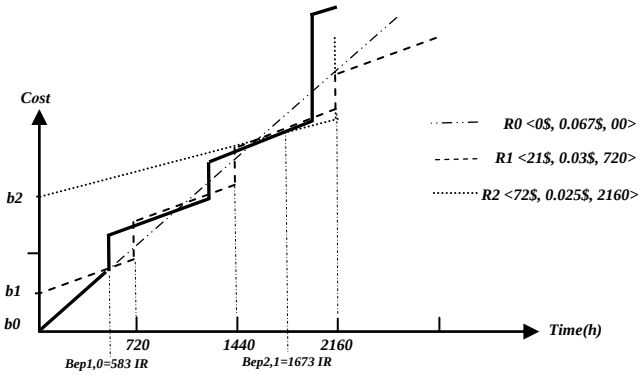


Figure 6. A multislope rental instance with 3 slopes. The thick line indicates the accumulated cost when using *bepa*.

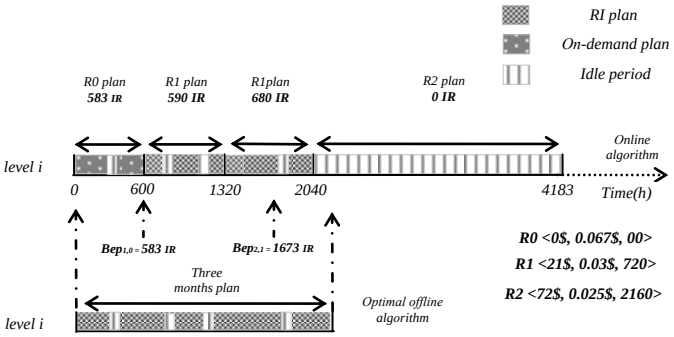


Figure 7. Illustration of algorithm *bepa* for *level i*

serve the demand instead, which would have lowered the cost, so as a fixe to this "mistake" , we reserve R_1write after a number of $Bep^t_{1,0}$ instance requests(IRs), and of course $Bep^t_{1,0} > d_1(583 < 720)$, otherwise using R_1 is not cost effective and we have to retain R_0 plan. Once R_1 is over we check if we reached the *break-even point* $Bep^t_{2,1}$ while using R_1, in this example it didn't happen during the first R_1, so as mentioned previously, we have to stick with the current offer, which is R_1, but if a certain condition is verified. Now imagine if there were no instance requests during R_1, and our bepa decided to reserve R_1 for the second time, surely bepa will lose a lot of money, so there has to be a condition based on which we can choose the right offer to reserve. In bepa at *step 16* we defined a condition where we compare between the *budget* spent in previous reservations plus the effective instance requests cost $r_c * D^t_c$ minus the cost of $idle$ period during the current R_1 offer, and $Cost(R_1)$. So, if $budget > Cost(R_1)$, even if there were no instance requests during the second R_1, we are sure that this second reservation of R_1 is bounded and its cost is lower than what we have spent previously. In Fig. 6, this condition is indeed verified, $budget = 114\$$ and $Cost(R_1) = 43\$$, so bepa proceeds to a second reservation of R_1 at time 1320. If the condition $budget > Cost(R_1)$ fails, then we have to find a new $\{R_j\}_{0\le j\le i}$ where $Cost(R_j) < budget$. Now that we rent R_1 for the second time, we check if we reached the *break-even point* $Bep^t_{2,1}$ while using the current R_2 in order to move to R_2, in this example $Bep^t_{2,1} = 1314$, so we definitely exceeded, so once R_1 is over we have to rent R_2 offer, but again if a certain condition is verified. We definitely spent in previous reservations a cost more than $Cost(R_2)$, so even if there were no instance requests during R_2, we still can bound the cost incurred by R_2, by finding a lower bound for the

interval [0,1314] . However, if the number of instance requests during the last R_1 is slightly above $Bep_{2,1}^{t} - D_2^t =$ 14*Bep*2,1 - *D(2,t) = 14 (ed = 4546),* Cost*bepa* will incurs additional fees *ED*r1* (i.e., (2023-1314)*r1), since the optimal offline algorithm would have used on-demand plan for the 14 instance requests, thus the additional fees *ED*r1* will not let us to bound Cost *bepa.* So as a solution to this problem, we have to ensure that there is enough new demands, so that nd > ed, and thus the interval [0, 2036] can be bounded. If this condition is verified we can reserve R_2 (it is the case in our example), otherwise we have to find another Rj 0<j<I where Cost(Rj) < *budget,* so that Cost*bepa* can always be bounded.

Theorem: For any demand sequence, bepa is $\frac{2}{1-\alpha}$ competitive

Proof: let $Cost_{bpa}(t) = Cost_{bpa}(t^-) + Cost(rem)$

We know that $Cost_{bpa}(t^-) < \frac{1}{1-\alpha} Cost_{opt}(\mathrm{t})$

And since we do not rent the last R_i offer until

$$Cost(rem) \leq Cost_{bpa}(t^-)$$

Then $Cost_{bpa}(\mathrm{t}) \leq \frac{2}{1-\alpha} Cost_{opt}(\mathrm{t})$

II. Deterministic Online Algorithms with short term prediction

Previously, we analysed algorithms with either full knowledge of future workload (optimal algorithm strategy), or with no workload prediction consideration, but in this section, we focus on a new line strategy, which takes in consideration short term workload predictions. This can be helpful to reduce the overall risk introduced by our online algorithms, since we can use the short prediction to enhance the reservation decisions by avoiding long and non-profitable plans while promoting small profitable reservation plans. Thus, bring our algorithms closer to the optimal strategy. Short term predictions can be easily computed and estimated during the first months of cloud usage, once the user have control and better understanding of how the backend code is performance in a cloud environment after a series of enhancement, short term predictions can be reliable at a certain point.

Let e_t be our estimated short-term prediction limit, that means that at any moment t, our algorithms know about the future workload in the next e_t duration, and let's assume that e_t is always lower than the smallest reservation plan available: $e_t \leq \{d_i\}_{1 \leq i \leq n}$, so we can get the most reliable prediction possible. In our simulation, we set $e_t \leq 15\ days$, as a realistic short-term prediction for new cloud users. Now, for our algorithms to be able to extend their decisions, we need to consider both workload history, plus future workload predicted.

The deterministic mda adaptation to short-term prediction: We start serving demands by using on-demand instances, unlike regular mda strategy, we do not have to wait until the user spends an equivalent budget to the smallest reservation offer available, but rather, we can predict the traffic demand for the next $t + e_t$ duration, and then decide earlier whether or not we should reserve an instance. This small change in the algorithm's decisions, saves many Idle reserved instances.

1. Initialization $t = 0$, $ReservationSet = \{\}$
2. If i existe such as $i = argmax(d_i |\ d_i \leq \sum_{t+e_t-d_{i-1}}^{t+e_t} Check(D_t > ResN_t))$ then
3. Reserve $R_i$offer, and add $R_i$to $ReservationSet$
4. For each $< R_i, rem_i >$ in $ReservationSet$ decrement di value and remove R_i from $ReservationSet$ when $rem_i = 0$
5. Lunch on-demand plan for the remaining instances
6. Set $t = t + 1$, go to *step 2*

In Fig 8 we applied this algorithm to the same demand curve used in Fig 5, while using 2 weeks as a short term prediction available at any time, we can see that both Idle reserved instances incurred in level 4 and level 1 disappeared (because of the 15 days' workload prediction, we know that the VM request will not last for all that duration, the algorithm omits the next RI offer, thus it falls back to the on-demand plan), while it has been reduced in level 3 and 2. The larger the e_t, the more we get closer to the offline algorithm. As we can see in Fig 9, we have more cost-effective strategy when e_t is larger (1 month). For instance, level 2 has no Idle reservation period, thus the overall VM provisioning cost has decreased dramatically. In the next sections, we will see how this algorithm can perform under small short-term prediction values.

The same construction can be applied to bepa over a family of short term prediction values, we skip the formulation part for simplicity reasons. More benchmark evaluations can be found in the next sections.

III. Simulation

In the previous sections, we have analysed our proposed algorithms in terms of cost performance regarding the competitive analysis. In the remaining, we simulate our algorithms in a real use case, using large dataset of cloud users

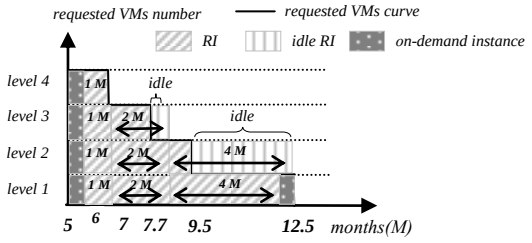


Figure 8. Applying mda with a short term prediction (0.5 month ~ 2 weeks) to the same demand curve used in Fig. 5

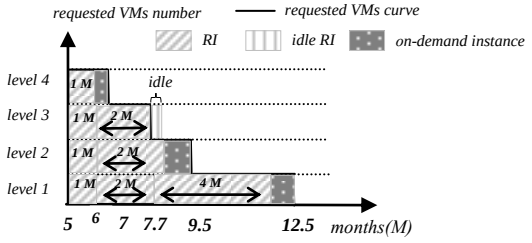


Figure 9. Applying mda with a short term prediction (1 month) to the same demand curve used in Fig. 5

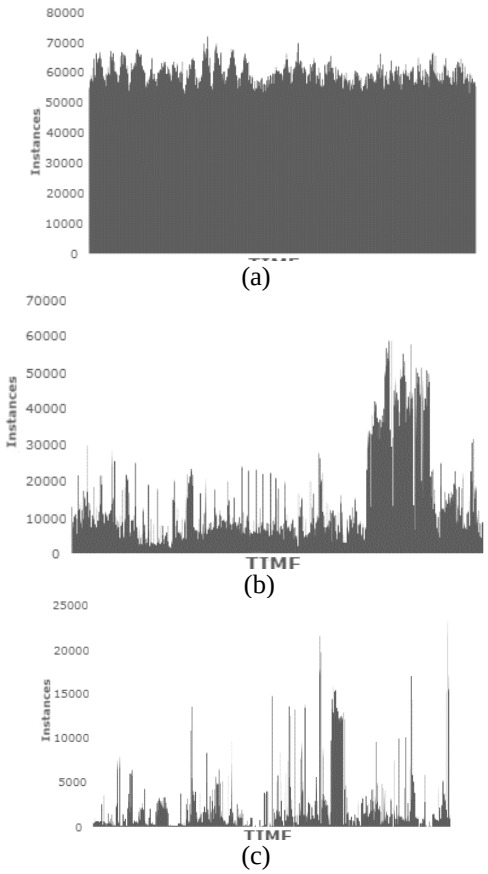


Figure 10. Users demands filtred by the standard deviation: (a) refers to a stable workload ($0 \leq \sigma/\mu < 1$), (b) refers to a medium worklad ($1 \leq \sigma/\mu < 5$) and (c) refers to a sporadic workload ($5 \leq \sigma/\mu$)

A. Dataset Description and Preprocessing

We were not able to find any public information about any cloud provider's usage, because mainly it's confidential and could damage their reputation, in case some repeated downtime or degraded performance were found in their public dataset, so we are currently bound to using the google cluster-usage traces [17], which is not a public cloud though, but it reflects some google services usages, and some google engineers computing usages. Overall, it can be fairly used to perform benchmarks between algorithms or strategies against more than 930 users contained (CPU, memory, disk, etc.) in Google's data trace over 29 days in May 2011. The data trace represents a cluster computing workload of more than 11K instances, with more than 50GB of csv resources

B. Dataset adaptation to a cloud enviroment:

Dataset adaptation is not an easy task, we need to accurately estimate home many instances each user requires if it meant to be run in a public cloud, so tasks scheduling is important along with machines/clusters computing adaptation. Basically, we had to consider the following constraints in order to achieve an accurate adaptation to a cloud environment:

- *RAM VS CPU usage*: We consider leasing a new virtual machine once it's either the RAM or the CPU reach the threshold of the current host machine (network usage should be considered too, but it's not available in the dataset)

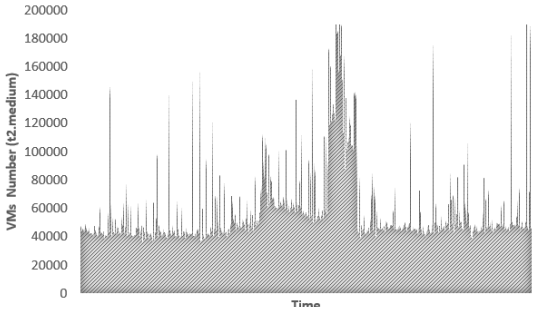


Figure 11. The demand curve of User 401(with stable workload) in Google clusterusage traces[14], over 1 month, adapted to a t2.medium AWS EC2 instance

- *Parallel tasks of the same jobs*: We consider tasks running in the same time even if they belong to the same job as separate tasks, therefore they are duplicated in the tasks queue, and incurs workload to the cloud infrastructure.
- *Tasks with constraints*: Computational tasks that cannot run on the same server in the traces (e.g., tasks of MapReduce) are scheduled to different instances

In the end, we obtain a demand curve for each user, indicating how many instances this user requires in each hour. Fig. 9 illustrates such a demand curve for a user.

C. Dataset duration

Google data trace span only 30 days, so we have to proportionally decrease the on-demand billing cycle, we choose one hour to be equal to one minute, and the reservation offer of 1an becomes $24h * 365j = 8760 \text{ min} \sim 6days$. Also, the dataset total duration becomes: $29j * 24h * 60 \text{ min} = \frac{41760\ min}{8760\ min} = 4.76\ years$. The break-even is scalled down to 28 instance-hours.

D. User Classification:

In order to evaluate our online algorithms, when we have stable, medium and sporadic on demand traffic, we sort the 930 users into 3 groups, based on the traffic fluctuation (standard deviation σ and the mean μ).

Group 3 represents users with a high sporadic traffic (i.e., $\sigma/\mu \geq 5$). In Fig. 10 we can see clearly that these users have a small means, therefore, they should use on demand instances as a VM provisioning solution.

Group 2 this group represents users that have a medium traffic workload, with $1 \leq \sigma/\mu < 5$. In Fig 10 we can see that they in the second place as the most dominant users, they actually need a dynamic provisioning strategy (i.e., both on-demand and reservation plans should be considered).

Group 1 it represents the most dominant users type, they have a stabilized workload with $0 \leq \sigma/\mu < 1$, with a large mean, they need to be served using reserved plans only.

E. Pricing Model:

In this simulation, we adopt the pricing of Amazon EC2 of a t2.medium instance from the marketplace (Jan 2017):

- On-demand hourly rate: 0.052$/h
- One-month reservation offer: 0.0468$/h
- Three months reservation offer: 0.0416$/h
- Six months reservation offer: 0.0345$/h
- One-year reservation offer: 0.0286$/h

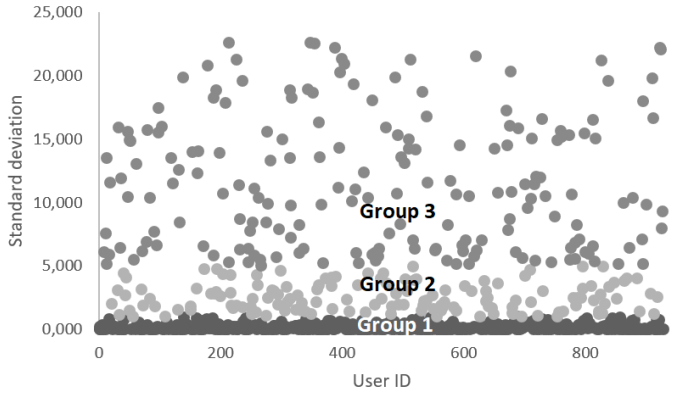


Figure 12. User demand statistics and group division

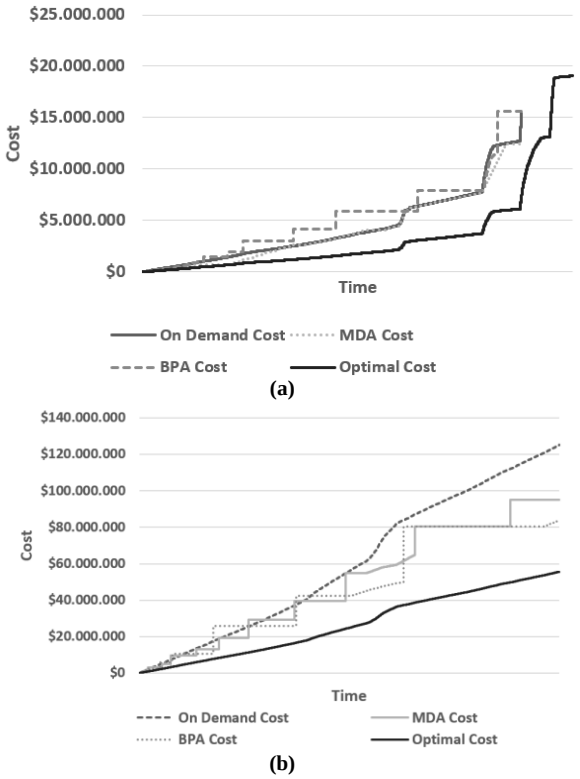


Figure 13. Cost performance of online algorithms without a priori knowledge of future demands: (a) refers to a sporadic workload, (b) refers to a stable workload

- Three years reservation offer: 0.0231$/h

F. Online Algorithms

We start carrying tests of our online algorithms, without any knowledge of the future workload.

Benchmark Online Algorithms: Our benchmark is performed like the following: we start by evaluating our two online deterministic algorithms against an on-demand pure plan strategy, where each user uses only on-demand instances, this strategy is simple and straightforward, no complexity involved, though it's widely used by most cloud new users, especially for start-ups that has been using cloud for less than 2 years (i.e., they don't have an accurate start-off estimation of how many servers they will need for their business, plus it roughly depends on their backends code quality and complexity). Our online algorithms take an All-on-demand strategy whenever the workload ends before the break-even time. Our second benchmark is when all resources are reserved from the beginning, this typical for old cloud users that have an accurate estimation of how many instances they need to run their business, so they start-off with a major resources reservation to reduce computing cost. In the following we test these two algorithms alongside with our online algorithms for each google users group.

Cost Performance: We see in Fig. 8a that when applied to Group 1 and 3, our deterministic online algorithms realize significant cost savings compared with the two benchmarks. In particular, when switching from All-on-demand to our deterministic algorithms, we can achieve a 24% and a 33% saving with both mda and bepa algorithms respectively when tested against a stable workload Fig. 11a, while the optimal algorithm (*which is not realistic, we just use it for benchmark, it refers to the lowest possible VM reservation cost per hour, regarding the upfront cost or the idle periods*) can save up to 30% and 35% when compared against our deterministic algorithms respectively.

In the other side, in Fig. 11b, bepa as expected is not performing well against extreme(sporadic) workload, and its underlying cost is mostly above all other algorithms. We can see at first that the two-deterministic algorithm are performing with caution, no reservation during the first months until the 6th, both mda and bepa make a reservation during the same month but with different reservation contract, bepa algorithm go for a couple of 6-months reservation offers, which is very risky regarding the workload's nature, while mda starts making a few 1-month reservations, it seems though to equals the All-on-demand algorithm expenses (which is the best choice when it comes to highly sporadic workload) because it simply doesn't take too much risk, and therefore it remains very close to the All-on-demand algorithm. The optimal algorithm though makes significant saving over all algorithms, up to 50% of cost saving against All-on-demand and mda, and up to 70% when compared to bepa which is the worst performing in this group.

In the flowing section, we switch back to the performance of our enhanced algorithms considering the short term perdiction reliability. We omit the part on how we predict short terms of workload periods, since it's roughly related to the user's business model, and we focus on the future VMs demands adaptation with the google data-trace. We set the same linear down scale strategy used previously to adapt two short term prediction categories: **1 month** equals *12 hours* and **3 months** equals *36 hours*. For each one of these categories, we roll our online algorithms without any knowledge of future workload for cost benchmark and verify if we can further reduce the cost of VMs acquisition with this strategy in case of a stabilized or a highly sporadic traffic, and how far it can be true.

In Figs 14 and 15, we normalize all costs to mda and bpa respectively. We can see that in all chart lines, cost have been reduced effectively for both short term prediction categories (1-month and 3-months). Having more prediction timeline helps definitely make better reserving decisions to effectively avoid useless VM reservations when the workload goes down, but it doesn't mean that you can save cost over strategic VM reservation offers. Reserving the

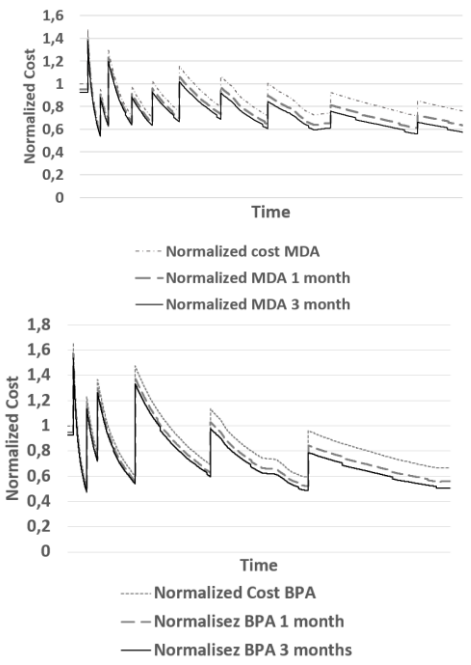


Figure 14. Cost performance of online algorithms with different short-term prediction windows when simulating a stabilized vm demands. All cost are normalized to the mda and bpa strategies respectively without any future information: (left) refers to mda benchmark, (right) refers to bpa benchmark

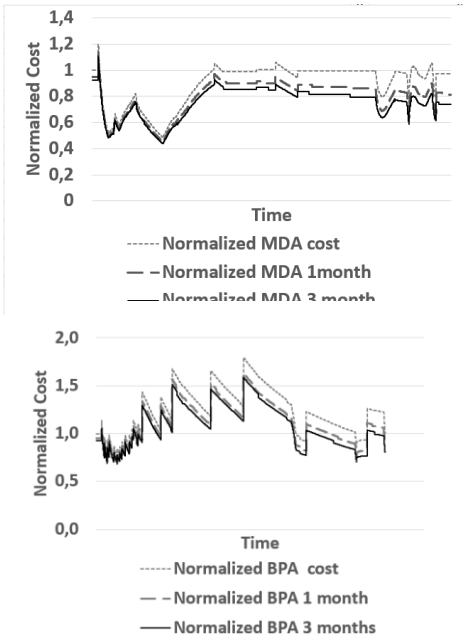


Figure 15. Cost performance of online algorithms with different short-term prediction windows when simulating a high fluctuating vm demands. All cost are normalized to the mda and bpa strategies respectively without any future information: (left) refers to mda benchmark, (right) refers to bpa benchmark

right offer at the right time and thus saving cost within the same decision needs a better algorithm. In other words, the short-term prediction strategy only helps to reduce the lose accumulated by all bad reservation offers made by the algorithm, moreover, the reduction made between different short-term prediction categories is not linear. For instance, in Fig. 14 (a), we can see that the difference between knowing a 1-month and 3-months of future demand is not proportional to what we can save when running mda with 1-month over mda with no a piori knowledge, the benefits are decreasing. The same thing applies to bpa strategy, in which the 3-months prediction overall cost was very close to the 1-month strategy beforehand.

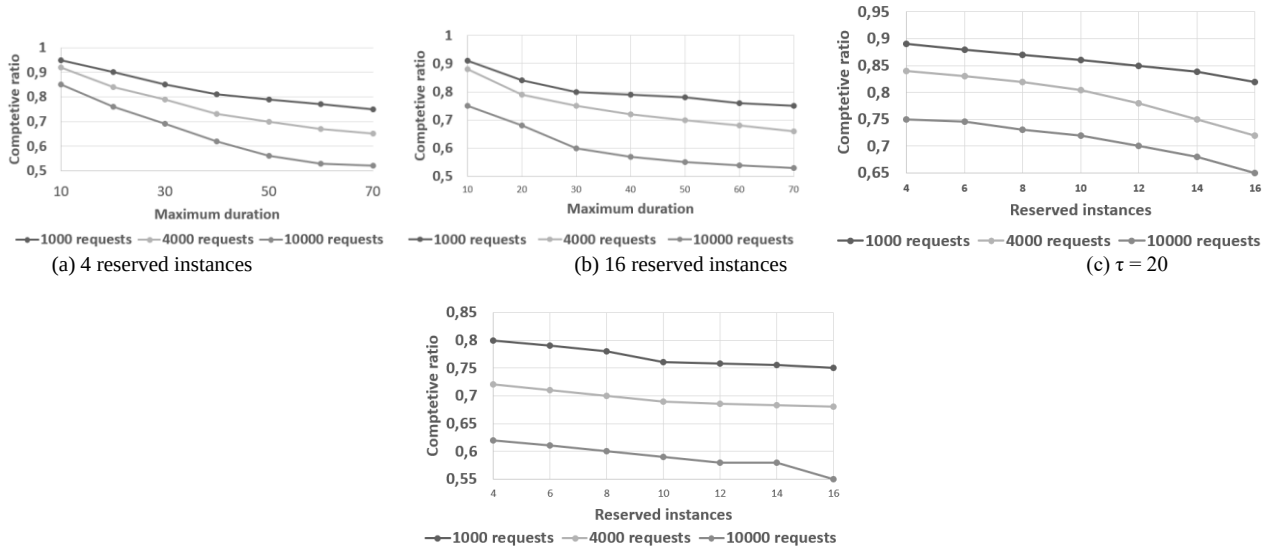
In the other hand, we can see in Fig. 15 an improvement of more than 6% over the 1-month strategy. Since this is a sporadic workload simulation, and because of the high number of bad reservation offers being saved by the algorithm with a larger prediction window, we can see this improvement in the chart-line, more knowledge we have about the future workload (i.e., longer prediction period), more bad reservation offers are cancelled. But again, at a certain point, even with a high prediction window, margin benefices go down, leaving no space for more improvements. We note also that, with certain users in the data-trace, having an extreme sporadic demand, the on-demand strategy is very close to the optimal strategy, which means, the all x-months prediction strategies are irrelevant, and have no improvement over the cost acquisition.

In this section, we evaluate the competitive ratio of our online algorithms regarding dynamic instance requests (1000, 5000, 10 000 requests). Let's assume that we have a m4.large RI type, low usage for one-year commitment. The hourly rate of an on-demand instance in the US East region is \$0.1, and the upfront fee for one-year commitment is \$61, while the hourly rate goes down to \$0.034. Let's assume also that the owner is willing to sell no more than 200 hours of his RI, and his asking for an admission price around [\$0.002, \$0.026]. Basically, the seller has to define a couple of parameters, like the maximum duration τ and c. So, let's assume that c=2 and within a time interval $[0,T]$, we randomly setup a set of requests $\{r_i\} = \{(b_i, k_i, s_i, t_i)\}$, where s_i is a random value in $[0, T-1]$, t_i is also chosen from $[0,\tau]$, k_i is chosen from $[0,m]$, and b_i and k_i depend on each other based on the bid price point.

In the following, we evaluate the competitive ratio of our online algorithms while considering all previous parameters. We initialize a fixed set of reserved instances and dynamically change the maximum duration value in order to study the τ value, then we initialize the maximum duration with a fixed value and vary the number of RIs. We can see in Fig. 16 all different variation of the competitive ratio under different instance requests scenarios.

Fig. 16a and Fig. 16b evaluates the maximum duration variation. When this value goes up, the competitive ratio decreases. This confirms our previous theoretic analysis. Fig. 16c and Fig. 16d evaluates the variation of the number of reserved instances. Both the competitive ratio decreases and the number of reserved instances increase. This is due to the assumption conditions, more instances are being requested with a low bid value, therefore the number of reserved instances increases. We can see that the competitive ratio of *mda* algorithm is reaches 45 % in all three scenarios, while our second *bpa* algorithm reaches 55%.

Next, we benchmark our online algorithms against two additional algorithms:



(a) 4 reserved instances (b) 16 reserved instances (c) τ = 20

(d) τ = 40

Figure 16. Benchmarking the comptetive ratio of our online algorithms against alg A and alg B in different setups.

1) alg A: where each user opts for the RI offer that has the maximum bid price available.

2) alg B: where each user opts for the RI offer with the maximum bid frequency value available, i.e., b_i/t_i.

Assumption is not used by any of these two algorithms. We can see in Fig. 17 that our online algorithms perform well compared to our two benchmarks.

For the next simulation, we are using the same google data trace used in the previous benchmark. Each job contains a set of tasks, either with the same or different resource requirement. Because of the hourly billing nature, we consider only long jobs that are running for more than 1h from google trace, they are about 39k jobs, so we can evaluate the cost saving performance of our online algorithms. In order to operate an accurate simulation, we start by concluding how many instances are required per each job if it were to execute in a real cloud data center scenario. We proceed with the following adaptation of the google cluster dataset to accurately schedule different single and parallel task jobs: 1- All single task jobs are gathered into one single instance until one of the core VM resources gets exhausted, then we move to another new instance. 2- For the parallel task jobs, in most scenarios, these are MapReduce based tasks, so they should be scheduled into different instance type. We note that in each second, most google cluster jobs need around 100 instances. So, we assume that each seller has more than 100 instances, say 200 RIs, and looking to sell about 1 month of usage from each instance. In Fig. 18, we plot our benchmark against algorithms A and B, and we can see that *bpa* and *mda* are performing about 15% and 20% less than our close competitor (B algorithm).

We omit the scenario where we benchmark the maximum duration, because many jobs in the data trace stay in the active state less than 15 hours, which makes the total saving over our online algorithms almost the same with either a 1-month, 2-month or a 3-month RI duration.

In this section, we introduce a broker service that sells on demand instances to the end users, with a reduced price compared to the full AWS on demand instance prices. We analyse the performance of this broker if it were to run our online algorithms as an instance reservation strategy. Then we benchmark our algorithms against heuristics algorithms that are usually used by broker services.

For a correct simulation, we combine instance requests of all users who belong to the same demand *group* (demand fluctuation) into one single group. This is because we get many instance hours wasted if each user makes his own instance purchase, for more clarification,

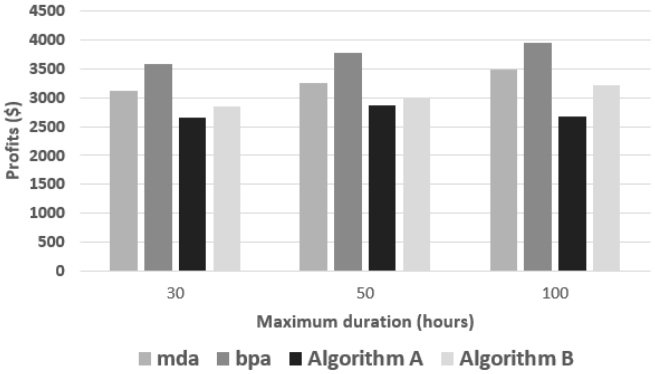
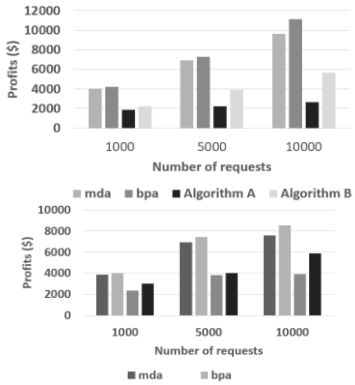
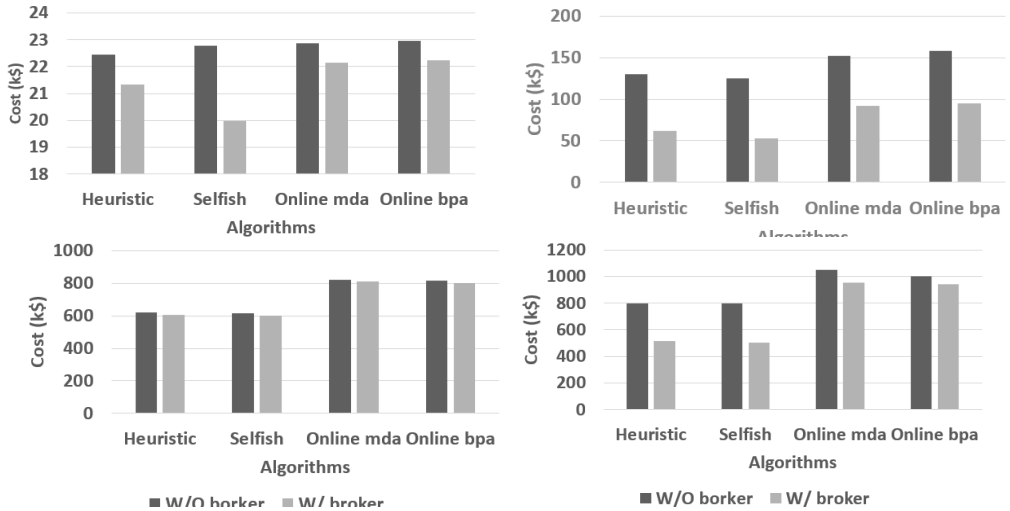


Figure 17. Benchmarking the overall profits among 4 algorithms using Google datatrace, as input $m = 200, T = 720h$



(a) τ = 20 (b) τ = 40

Figure 18. Benchmarking the overall profits among 4 algorithms using Google datatrace, as input $m = 5, T = 200h$



(a) Group 1: high/sporadic workload (b) Group 2: Medium workload (c) Group 3: Low workload (d) Group 4: All workload = 40

Figure 19. Combined workload cost considering broker usage in different workload classes

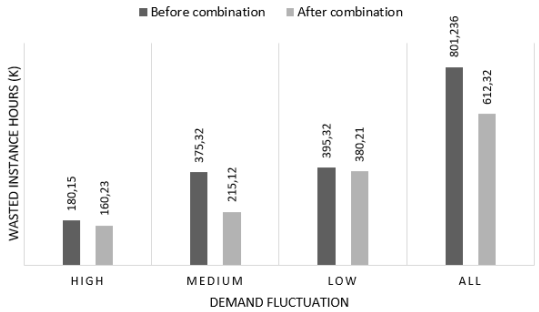


Figure 20. Combined workload reduces wasted computing hours.

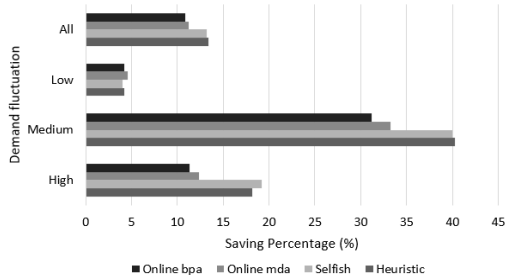


Figure 21. Benefits of using a broker service with a combined cost saving for different workload classes.

in Fig. 20 we plot the wasted instance accumulated for each demand fluctuation group. For instance, we compare the average instance hours consumed by a group 2 user, if it were to be purchased directly from AWS, versus the average instance hours consumed by all group 2 users combined. We run through the remaining 3 demand groups, and we visualize the cost saving percentage accumulated in each scenario. The medium demand fluctuation seems to be the one that is making more profits than others, while we expected that the highly demand fluctuation makes significant profits, it turns out that it's wrong, most *group 3* users have a low instance usage, therefore there was not enough instance requests to be combined, so the overall wasted instance hours was small.

Now, we have to evaluate this combined demand instances strategy if it were to be operated by a broker service and see how our online algorithms perform against a classic heuristic algorithm. In Fig. 20 we plot the accumulated cost for both the single user and the broker service running the same reservation strategies for each single demand fluctuation group. While Fig. 21 shows the overall profit when we use a broker for each particular group. We can see that the profit can reach 15% when we combine all user groups. The same result can be seen in Fig. 19d, where the broker saved more than 73k$ with all users combined. We note that the broker profits vary depending on instance requests fluctuations, for instance, the broker realized over 40% profits with a medium traffic, while the smallest profits were reported with a stabilized traffic (~5%), this can be seen in Fig. 19c. This is due to the fact that users with stabilized traffic are already using RIs for most of their demand instances, therefore the broker's spending is very close to the user's spending. In Fig. 19b, we can see that the broker makes good profits through instance demand combination, thus better using reductions of RIs. However, in Fig. 19a, even with the combined instance requests, the profits are not optimal, the broker still don't have enough instance requests to purchase enough RIs, thus making less profits than a medium user traffic. But we still get a 15% ~ 17% cost reduction due to instance demand combination.

Now, we benchmark the cost of our online algorithms against selfish and heuristic strategies. Fig. 19 shows that the selfish strategy is the most successful one, then comes the heuristic strategy, and finally our online algorithms. This is due to absence of the forthcoming instance requests. However, in Fig. 19a, all 4 algorithms have similar curves with a sporadic workload, this because most instances come from an on-demand plan, only a few are effectively RIs, thus all strategies decisions become less important.

From all broker's performance results, we can say that more than 75% of users that belong to the medium workload demand, save more than 30%, while the same broker can save more than 25% in favour to 70% of all users combined. We also report that there is a certain limit (i.e., 50%) on the maximum profit that each user can make, also we found that with both our online algorithms, more than 40% of users made a profit around 30% (this the scenario where the broker earns the most), and only a small

group made less than 4%, that's because they have requested only a small number of instances of the entire demand, thus the broker have to charge them with a cost per hour very close to the cloud pricing catalogue. There are many strategies to satisfy this kind of users, where the broker has to over compensate for their sporadic traffic at the expense of other users belonging to the medium workload group, but these strategies are out of scope of this paper.

In this section we analyse both benefits and challenges of a cloud broker system that haven't been discussed in the previous evaluation. First, all cloud providers have special pricing plans for large business, beyond the reservation plan, which means the cloud broker can make much more benefits when contracting with a cloud provider, for instance AWS offers a 20% discount for heavy ec2 usage apart from RI offers, thus cloud brokers are easily qualified for these large contracts.

Second, besides the saving of computing resources, the broker can reduce the storage and bandwidth allocation, thus reducing its usage, the cost of combined resources is always cheaper than the cost of allocating separate services from an IAAS cloud provider.

Third, the cloud broker can help start-ups with small business to overcome the fear of cloud migration and expensive computing bills, since most instance acquisitions are made through RI offers, they are much cheaper, cost-effective with no risk, lasting for years as a business plan model.

However, the broker has several limitations and disadvantages, either related to the server maintenance or operating the service. First, the broker's partial profit depends heavily on the pricing catalogue of each cloud provider. Also, starting a new OS system for a new user on the same VM is billed as a new hour cycle (e.g., AWS, but not all cloud providers have this limitation), thus not saving much over user instances mixing strategy. Actually, in our evaluation, we can see that even when omitting the instances mixing strategy, we can still save 8% over the total expenses.

Second, most high traffic sites don't have control about when workload picks can happen, so the broker estimation and online decisions may not work properly, in fact, it could be a disaster in some scenarios, and the broker will have to bump off wasted RI at a bad price. But after all, these users will have to deal with the same circumstances when acquiring instances directly from IAAS cloud providers.

Third, in our previous evaluation, we assumed that users can benefit from all the cost saving realized by the broker, so the broker will not find any difficulties in acquiring new customers and retaining old once. But in fact, the broker has to take a portion of the reservation benefits or through a percentage over total revenue, so there will be not much room for customers that already have a reduced computing cost. For instance, a large cloud customer has already a direct contract with the cloud provider that is probably in the same pricing level as the broker. So, only small business would be interested in using a cloud broker service, in addition to that, the broker has to drop prices at the lowest level to seriously attract new clients, and retaining old customers, because if the price point is close to the cloud's pricing plan, all customers will prefer using the cloud directly since it has other interface integration with other services like storage, load balancer, web application firewall… etc.

Fourth, maintaining a cloud broker service is not an easy task, especially if it serves many small business, the broker should be prepared to a large number of case issues (so it needs a dedicated team for customers support), many customers will ask for an integration with other cloud services within the broker's API (so it means a dedicated team for continues development and testing), also it is very difficult to provide a correct *SLA*(service level agreement) when the broker uses different cloud providers in the same to time to serve customers.

In this section we run a new set of experiments with different website traffic scenarios. We schedule AWS reserved instances at 4 different situations in order to evaluate the performance of our online algorithms:

1. Scenario 1: *Website with an average audience traffic but with small pick load time*. The average request per second (RPS) rate of this site is around 300 and it's twice during workload pick times. The annual revenues forcast of this site should be around 2M$
2. Scenario 2: *Website with an average audience traffic but with high pick load time*. This is similar to scenario 1, except that the pick load is 8 times superior than the regular workload. The annual revenues forcast of this site should be around 3.5M$
3. Scenario 3: *Website with a large audience traffic but with small pick load time*. The average incoming requests rate of this site is around 300 requests per second (RPS) and it's twice during workload pick times. The expected annual revenues of this site should be around 5M$
4. Scenario 4: *Website with a large audience traffic and a high pick load time*. Again, this is similar to scenario 3, except that the spike load is 8 times superior than the regular workload. The annual revenues forcast of this site should be around 7M$

For simplicity and reference we choose t2.small as the smallest AWS instance size available. Let's assume that the

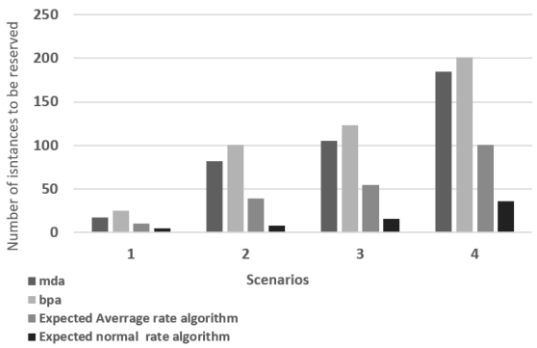


Figure 22. Number of reserved instances in each scenario.

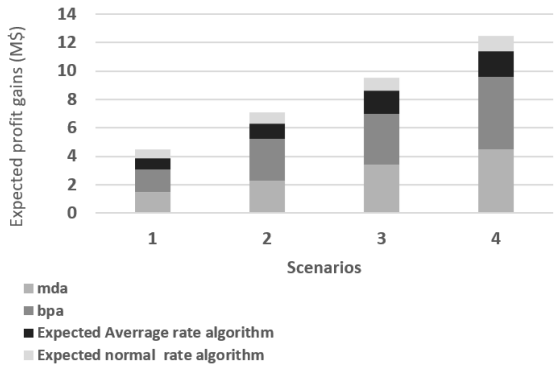


Figure 23. Expected benefits when using online algorithms versus short term prediction algorithms for each scenario

maximum number of instances that can be purchased at the same time is 20.

Web sites are mostly made of several backend/server layers, each with a specific goal, in order to get a fair and correct benchmark, we set the overall benefits of executing a single request as the same for all sites backend layers, besides, we set the computing needs of each request as the mean value of several request types combined, while considering the targeted backend layer family. For instance: if a request of type *A* needs 0.02 s and 0.03 s for processing during the first and the second backend layer respectively, and a request of type *B* needs 0.01 s and 0.04 s for processing during the first and the second backend layer respectively, and considering that most (80% of workload) of requests belong to type *A*, then, the mean value of a single computing request would be $(0.02 + 0.03) * 0.8 + (0.01 + 0.04) * 0.2 = 0.05\ s$. With this last assumption, we can say that the more workload we get, the larger is the benefits of the site.

We assume that the downtime of AWS instances is less than 99.99%, that's around 12 minutes a month of unavailability, the penalty of such infraction is about 12k$.

In Fig 5, we can see the amount of RIs in each scenario, we were able to manage good margin benefits with our online algorithms compared to the previous classic strategies. We can see a 25% increase in the RIs ratio compared to an average arrival rate strategy. We can see also a 13% increase of RIs when using bpa strategy over mda in both scenarios 2 and 4, this is because of the sporadic nature of the traffic, and the fact that we need an algorithm that is ready to take much risk to take advantage of all high pick load times and further increase the margin benefits. However, in scenario 1 and 3, mda remains the best strategy to use, since in both scenarios, the traffic is stable with small pick load times, that have small impact on the requested computing resources. The annual profits benchmark result is plotted in Fig 3.

Based on the presented results, we can conclude that our online algorithms can generate important cost saving regarding AWS computing investment, especially when used against a high traffic workload with high pick load time, we can see this result clearly in both scenario 3 and 4, while it remains fairly profitable when using mda for a traffic with small pick load time.

IV. Conclusion and Future Work

In this paper, we extended the work of Wei Wang et al. in the case where we have multiple reservation offers. Firstly, we proved that this problem is indeed NP-hard, and we proposed two practical online deterministic algorithms that incur no more than $1 + \frac{1}{1-\alpha}$ and $\frac{2}{1-\alpha}$ respectively, compared to the cost obtained from an optimal offline algorithm. Then we developed two other short-term prediction algorithms that further improves the competitive ratio. We focused on a large-scale simulation of previous algorithms over the Google cluster-usage traces. We evaluate our strategies regarding dynamic instance requests, reserved duration, dynamic bid price, combined versus separated workload, competitive ratio, and more. Over 30% of computing expenses can be saved when using our algorithms, while 40% when customers go through a cloud broker service. One of the issues that we have not discussed is the probability-based algorithms along with the combinations of different cloud providers offers (i.e., Rack space Hosting, Google App Engine, Amazon Ec2…). We are confident that these combinations could further reduce the instance acquisition cost.

References

[1] Azar, Y., Bartal, Y., Feuerstein, E., Fiat, A., Leonardi, S., & Rosén, A. (1999). On capital investment. Algorithmica, 25(1), 22-36..

[2] Lopes, R., Brasileiro, F., & Maciel Jr, P. D. (2010, April). Business-driven capacity planning of a cloud-based it infrastructure for the execution of web applications. In Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on (pp. 1-8). IEEE.

[3] Stage, A., Setzer, T., & Bichler, M. (2009, June). Automated capacity management and selection of infrastructure-as-a-service providers. In Integrated Network Management-Workshops, 2009. IM'09. IFIP/IEEE International Symposium on (pp. 20-23). IEEE.

[4] Bejerano, Y., Cidon, I., & Naor, J. S. (2000, August). Dynamic session management for static and mobile users: a competitive on-line algorithmic approach. In Proceedings of the 4th international workshop on Discrete algorithms and methods for mobile computing and communications (pp. 65-74). ACM.

[5] Damaschke, P. (2003). Nearly optimal strategies for special cases of on-line capital investment. Theoretical Computer Science, 302(1), 35-44.

[6] Wang, W., Liang, B., & Li, B. (2015). Optimal Online Multi-Instance Acquisition in IaaS Clouds. Parallel and Distributed Systems, IEEE Transactions on, 26(12), 3407-3419.

[7] Yao, M., Zhang, P., Li, Y., Hu, J., Lin, C., & Li, X. Y. (2014, June). Cutting your cloud computing cost for deadline-constrained batch jobs. In Web Services (ICWS), 2014 IEEE International Conference on (pp. 337-344). IEEE.

[8] Meyerson, A. (2005, October). The parking permit problem. In Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on (pp. 274-282). IEEE.

[9] Zhang, G., Poon, C. K., & Xu, Y. (2011). The ski-rental problem with multiple discount options. Information Processing Letters, 111(18), 903-906.

[10] "Roundup Of Cloud Computing Forecasts And Market Estimates, 2015," http://www.forbes.com/sites/louiscolumbus/2015/01/24/roundup-of-cloud-computing-forecasts-and-market-estimates-2015/

[11] "How Reserved EC2 Instances Work," https://skeddly.desk.com/customer/portal/articles/1348371-how-reserved-ec2-instances-work

[12] Uehara, R., & Uno, Y. (2004). Efficient algorithms for the longest path problem. In Algorithms and computation (pp. 871-883). Springer Berlin Heidelberg.

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

[14] “Google Cluster-Usage Traces,” http://code.google.com/p/ googleclusterdata/.

[15] Costa, C., & Santos, M. Y. (2017). Big Data: state-of-the-art concepts, techniques, technologies, modeling approaches and research challenges.

[16] Zhongda Tian, Shujiang Li, Yanhong Wang, and Bin Gu, "Priority Scheduling of Networked Control System Based on Fuzzy Controller with Self-tuning Scale Factor," IAENG International Journal of Computer Science, vol. 44, no.3, pp308-315, 2017.

[17] Gaizhen Yan, Ning Wu, and Zhicheng Zhou, "A Novel Non-cluster Based Architecture of Hybrid Electro-optical Network-on-Chip," IAENG International Journal of Computer Science, vol. 44, no.3, pp368-374, 2017.