Online Mechanism Design for VMs Allocation in Private Cloud

Xiaohong Wu, Yonggen Gu, Guoqiang Li, Jie Tao, Jingyu Chen, Xiaolong Ma

Online Mechanism Design for VMs allocation in Private Cloud

Xiaohong Wu[1], Yonggen Gu[1], Guoqiang Li[2,*], Jie Tao[1], Jingyu Chen[3], and Xiaolong Ma[4]

[1] School of Information and Engineering, Huzhou University, Zhejiang, 313000, China
[2] School of Software, Shanghai Jiao Tong University, Shanghai, 200240, China
[3] Institute of Computer Application Technology, Hangzhou Dianzi University, Zhejiang, China
[4] School of Information Management and Engineering, Shanghai University of Finance and Economics, Shanghai, China

Abstract. Resource allocation mechanism plays a critical role towards the success of cloud computing. Existing allocation mechanisms in public cloud is unsuitable for private IaaS cloud because they either cannot maximize the sum of users' value, or provide no service guarantee. This paper proposes a novel online, model-free mechanism that makes different allocations for flexible jobs and inflexible jobs. Users presenting job are incentivized to be truthful not only about their valuations for VM units, but also about their arrival, departure and the character of jobs (flexible or inflexible). We simulate the proposed online mechanism using data from RICC, showing that, compared with the mechanism which adopts same allocation method for all jobs, using our mechanism leads to high social welfare and percentage of served users.

Keywords: mechanism design, incentive compatible, resource reservation, greedy allocation

1 Introduction

With the development of cloud computing technology, *Infrastructure-as-a-Service (IaaS)* has gained popularity in recent years due to the flexibility, scalability and reliability. For a private IaaS cloud, the objective of resource allocation is to maximize the efficiency of resources. That is, the IaaS private cloud provider needs to find an optimum resource allocation for all users in order to maximize the social welfare which is the sum of users value.

Existing allocation mechanisms in public IaaS cloud are pay-as-you-go and bid-based allocation. Pay-as-you-go is a first-come first-serve allocation mechanism which does not concern about the value of an allocation. In fact, the efficiency of an allocation can be improved if the cloud allocates VMs to users with higher valuation by knowing user-centric valuation. Amazon [1] has used bid-based mechanism in spot instance market to make up for this shortcoming, where users periodically submit bids to the provider, who in turn posts a series of spot prices. Users gain resource access, until the spot price rises above their bids. Thus, due to the dynamic changing of the spot price, it provides no service

guarantee to those jobs which should be completed by VMs during multiple time unit periods.

To overcome above shortcomings in existing mechanisms, we are motivated to design a new auction mechanism for VM allocation in private IaaS cloud which is presented as an *online greedy allocation with reservation*(OGAWR) mechanism. The OGAWR mechanism has three characteristics compared with the auction in spot instance. First, the auction in our online mechanism is carried out in each time unit as long as user comes, while the auction in spot instance is carried out in each period that includes multiple time units. Second, OGAWR mechanism will provide service guarantee, that is, each job which should be processed during multiple time units will not be terminated before it is completed. Third, especially, we use different allocation methods of VMs for the flexible jobs and inflexible jobs. *Flexible jobs* refer to those jobs that the users only care about whether they could be completed before their deadline, and the process details are ignored. For example, a finance firm has to process the daily stock exchange data for guaranteeing the trading in next day. Obviously, the finance firm only cares about whether the job can be finished before the next trading day, and does not care about how the job is processed. Contrarily, *inflexible jobs* refer to those jobs that must be processed continuously when they start to be processed.

The rest of paper is organized as follows. In section 2 we discuss related work. After formalizing the problem model in Section 3, in section 4 we design an online greedy allocation with reservation(OGAWR) mechanism, and analyze the properties of OGAWR mechanism in section 5. The experiment evaluation are showed in section 6. Finally, conclusions appear in section 7. Due to the lack of space, we omit proofs of lemmas and theorems; these can be found in the extended version [16].

2 Related Work

The resources allocation in cloud computing is an important topic because it is closely related to the revenue of both cloud users and providers. Many literatures have conducted the studies focusing on this topic, and there are two main lines of research for this problem. One of these investigates the VMs allocation by solving an optimization problem. These works focus on the optimization of object functions, but generally without considering any strategic behaviours among users (e.g., the VM allocation approach for spot markets in paper [9]). The other is game theory based approach to analyze and design a reasonable mechanism. For instance, a cloud resource allocation approach based on game theory [7] is proposed, and assumes that the allocation would start after all users submit their request. Combinatorial auctions are supposed to apply in VMs allocation in some literatures [8] [10] [11], and all these work only consider resource allocations in one time unit and restrict their discussions in a single offline auction period. However, in cloud computing, the cloud users arrive and leave randomly, so the statistic analysis and design based on game theory are not suitable for it.

Online mechanism is an important expansion of mechanism theory in the multi-agent and economics literature, generally applied in dynamic environment,

which is consistent with the environment characters in cloud computing. According to the research on online mechanism, there are two frameworks of research in this field. One of these is model-based approach which aims for developing online variants of Vickrey-Clarke-Groves (VCG) mechanisms [3] [4]. These works rely on a model of future availability, as well as future supply (e.g., Parkes and Singh [3] use an MDP-type framework for predicting future arrivals). The other is model-free approach which requires fewer assumptions, and makes computing allocations more tractable than the first one (e.g., the online mechanism in electric vehicle charging in [5]).

The online mechanisms have been used in cloud computing [12] [13] [14]. [14] only introduces an online mechanism framework for cloud resources allocation without detail allocation algorithm. The online mechanism in [12] is a resource allocation approach for batch jobs, and the value functions for users are continuous. Zaman et al [13] design a truthful mechanism that allocates the VMs to users by greedy allocation, and allows those allocated users continuously use those VMs for the entire period requested.

Based on those works in [12] [13] [14], we also aim to design an online truthful mechanism for VMs allocation. Further, we pay attention to the following points:

1. In our model, a user requests one VM for multiple time units to finish the job during the arrival-departure interval. According to the demand in process time, all jobs are classified into two classes: flexible jobs and inflexible jobs.
2. We choose different allocation methods for the two classes of jobs, and especially design a discontinuous resource allocation based on reservation-ratio for the flexible jobs, by which the distribution of workload of users can be adjusted at their arrival-departure interval and the total workloads processed in cloud will be improved.
3. We focus on all the users with single-valued preference. That is, each user could get a non-zero constant value brought by the job only if it could be finished completely.

3 Modeling and Notations

We consider a private cloud provider who provides only one type of VM instances, and the total number of VM instances is denoted by C. Consider discrete time periods $T = 1, 2, ...$, indexed by t and possible infinite.

An agent presents a user i who submits its job to the cloud randomly, which can be characterized by the 'type' $\theta_i = (a_i, d_i, l_i, e_i, V_i) \in \Theta_i$, where Θ_i is its type space. Here, a_i and d_i present the arrival and departure time of agent i, and l_i is its total computation workload, i.e, the job size. Assume that each agent requires at most one VM in one time unit. The workload l_i is the number of time units for which agent i requires one VM. The last component of θ_i is V_i, the value agent i obtains if its job is completed, and $V_i \geq 0$.

As described in section 1, the jobs are classified into flexible jobs and inflexible jobs. In order to distinguish the job classes, a character parameter e_i is used to point out the agent is flexible ($e_i = 1$) or inflexible ($e_i = 0$).

We define $\pi_i = (\pi_i^{a_i}, \pi_i^{a_i+1}, ..., \pi_i^{d_i})$ as the allocation for agent i. $\pi_i^t = 1$ if agent i is allocated one VM at time $t \in [a_i, d_i]$, otherwise $\pi_i^t = 0$. The allocation result for agent i is denoted by A_i.

$$A_i(\pi_i) = \begin{cases} 1 & if \quad \Sigma_{t\in[a_i,d_i]}\pi_i^t \geq l_i \text{ and } \pi_i^t \leq 1 \\ 0 & otherwise \end{cases} \tag{1}$$

Each agent i is characterized by a valuation function v_i defined as follows:

$$v_i = \begin{cases} V_i & if \quad A_i = 1 \\ 0 & otherwise \end{cases} \tag{2}$$

The challenge of the cloud provider is to make allocation decisions π^t dynamically while trying to maximize the sum of agents value. The problem is described as follows:

$$\begin{aligned} & \max \Sigma v_i \\ s.t \quad & \pi_i^t \leq 1 \\ & \Sigma_{i=1}^n \pi_i^t \leq C, t \in T \end{aligned} \tag{3}$$

4 The Online Greedy Allocation with Reservation

4.1 Description of Mechanism

In this section we design a model-free online mechanism for the above setting.

The number of idle VMs at time t is denoted by $s(t)$. The definition of greedy allocation is as follows.

Definition 1 *(Greedy allocation) At each step t allocate the $s(t)$ VMs to the active agents with the highest valuations.*

If all agents request one VM only for one time unit, greedy allocation with appropriate payment could constitute a truthful mechanism [2]. However, in the case of multiple time units demands, according to equation (2), whether agent i could get the value V_i is decided by all of its allocation in period $[a_i, d_i]$. That is, the value brought by one VM at some time unit cannot be decided at first, so greedy allocation for each time unit can not be performed. In order to maintain incentive compatibility, we extend the greedy allocation policy by allowing the system to reserve VMs for agents. By such allocation approach, the agent is not only allocated one VM at current time t but also reserved one VM for multiple time units in future period. We define unit valuation as the valuation of one VM per unit time, and it is expressed as V_i/l_i to each agent i.

Definition 2 *(Online greedy allocation with reservation(OGAWR)) At each step t allocate the $s(t)$ VMs to the active agents with the highest unit valuations, at the same time, make the VM reservation for allocated agent i during period $[t+1, d_i]$ if $l_i > 1$.*

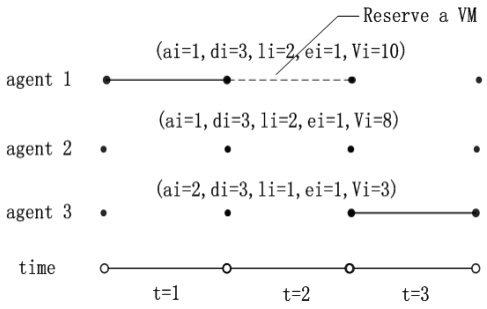

Fig. 1. An example for multi-time unit demand($C = 1$)

Consider an example with 3 time units and 3 agents in Fig. 1, where $\theta_1 = (1, 3, 2, 1, 10), \theta_2 = (1, 3, 2, 1, 8), \theta_3 = (2, 3, 1, 1, 3)$ showing the agents arrival, departure, job size, job class and valuation. Suppose furthermore that $C = 1$, and we sort the agents by their unit valuation V_i/l_i. Because agent 1 has the highest unit valuation at time 1, OGAWR method would allocate the VM to agent 1, and reserve one VM for it. Since it is a flexible job ($e_1 = 1$), there are two choices for reserving: reserving at time 2 or at time 3. In Fig. 1, the VM in time unit 2 is chosen to reserve for agent 1, so there is no idle VM to be allocated at time 2. At time 3, although agent 2 has higher unit valuation than agent 3, the VM is still allocated to agent 3, because agent 2 has no sufficient time to finish the job at that time.

It is worth to note that OGAWR might not be performed in some cases. That is, an agent with highest unit valuation cannot be allocated although there is sufficient time from departure for process. In the above example, suppose that the VM in time unit 3 is reserved for agent 1 at time 1. In that case, at time 2, although agent 2 has higher unit valuation than agent 3 and there is sufficient time to process, agent 2 still cannot be allocated. The reason for this result is that the supply in future is less than that in current time. Therefore, the OGAWR can be realized only if it makes 'non-increasing reserving'.

Definition 3 *(non-increasing reserving) Non-increasing reserving refers to a class of reserving schemes which always satisfies* $s(t) \leq s(t+1) \leq s(t+2) \leq ...$ *after allocation at each time* t.

In OGAWR mechanism, the agent participating the allocation at time t satisfies three conditions:(1)It arrives before time t. (2)Its departure time is longer than $t + l_i - 1$. (3)It is still unallocated. The OGAWR mechanism consists of allocation rule and payment rule described as follows.

- **Allocation rule** At each time t, it makes allocation as follows.
  Stage 1 *Greedy allocation*: Allocate the $s(t)$ VMs using greedy allocation, breaking ties at random.
  Stage 2 *Non-increasing reservation*: Make non-increasing reservation for agents who are allocated in stage 1 if necessary. If one VM is reserved for agent i at time k, $\pi_i^k = 1$.

Let $\theta^t = (\theta_1, \theta_2, ..., \theta_n)$ denote the set of agent types participating the allocation at time t, and π^t denotes the decision policy at time t. The mechanism makes a sequence of allocation decisions $(\pi^1, \pi^2, ...)$, and π^t includes all those agents allocated at time t.

– **Payment rule** We design a critical payment which is equal to the critical value for allocated agents, and the definition of critical value is as follows. Given type $\theta_i = (a_i, d_i, l_i, e_i, V_i)$, the critical value for agent i is defined as

$$V^c_{(a_i,d_i,l_i,e_i)}(\theta_{-i}) = \begin{cases} \min V_i' & s.t.\ A_i(\theta_i', \theta_{-i}) = 1, \\ & \quad for\ \ \theta_i' = (a_i, d_i, V_i') \\ \infty & no\ such\ V_i'\ exists \end{cases} \tag{4}$$

where $\theta_{-i} = (\theta_1, \theta_2, ..., \theta_{i-1}, \theta_{i+1}, ...)$.
We define payment policy $p_i(\theta)$ as

$$p_i(\theta) = \begin{cases} V^c_{(a_i,d_i,l_i,e_i)}(\theta_{-i}) & if\ A_i = 1 \\ 0 & otherwise \end{cases} \tag{5}$$

4.2 The Algorithm Design of OGAWR Mechanism

In this section, the algorithm based on the proposed rules for allocation and payment is designed. First, we introduce two reserving methods for inflexible agents and flexible agents respectively.

Continuous reserving: Continuous reserving is suitable for inflexible agents, which is similar to the allocation in MOVMPA mechanism proposed in paper [13]. If agent i wins the auction at time t, one VM will be reserved continuously for it in next $l_i - 1$ units. That is, $\pi_i^k = 1, k = t+1, ..., t+l_i-1$, if $\pi_i^t = 1$.

Discontinuous reserving based on reservation-ratio(Discontinuous reserving): This reserving method reserves one VM for agent i in next $l_i - 1$ time units with lowest reservation-ratio, and reserve the VM in earliest time unit if there are multiple time units with same reservation-ratio.

Reservation ratio denoted by $r(k)$ is the ratio of the number of reserved VMs to total capacity C at future time k expressed as $r(k) = s(k)/C$. Obviously, $r(k)$ is changed with time.

For inflexible agents, continuous reserving and discontinuous reserving both could be used. Since discontinuous reserving can adjust the distribution of users' workload, so in our mechanism we choose discontinuous reserving for inflexible agents. The steps of the allocation algorithm at time t are as follows:

Step 1 Sort all agents which participate the allocation at time t in non-increasing order of V_i/l_i.

Step 2 Allocate $s(t)$ idle VMs to $s(t)$ agents with highest valuation, breaking ties at random.

Step 3 Choose a suitable reserving method for each agent allocated at step 2. Continuous reserving is chosen if $e_i = 0$, and discontinuous reserving is chosen if $e_i = 1$.

Table 1. Allocation algorithm: *Allocate*

	Input: θ^t, S^t, t
	Output: S^t, π^t, A
1:	if $s(t) = 0$, goto end
2:	sort all $\theta^t \subseteq \Theta$ in non-increasing order of V_i/l_i
3:	$(\pi^t, A) = greedyallocate(\theta^t, S^t)$
4:	sort all $i \in \pi^t$ in non-decreasing order of d_i
5:	for each $i \in \pi^t$
6:	if $e_i = 1$
7:	$(\pi_i, S^t) = DiscontinuousReserve(l_i - 1)$
8:	else
9:	$(\pi_i, S^t) = ContinuousReserve(l_i - 1)$
10:	end if
11:	end for
12:	$S^{t+1} \leftarrow S^t \setminus \{s(t)\}$
13:	end

Define a status vector $S^t = (s(t), s(t+1), s(t+2), ..s(t+m-1))$ as the VM supplies in period $[t, t+m-1]$ before allocation at time t, where $s(t+k)$ is denoted as the supply at future time $(t+k) \in T$, and m satisfies $s(t+m-1) < C$ and $s(t+m) = C$. For computing critical value, we define $v_{-i,t}^{(m)}$ to be the m^{th} highest of unit valuations V_j/l_j from all agents j in θ^t, $j \neq i$. Then $v_{-i,t}^{s(t)}$, for supply $s(t)$, is the lowest value that is still allocated a unit, if agent i were not present not only at current t but also before t. Henceforth, we refer to $v_{-i,t}^{(s(t))}$ as the marginal clearing value of the idle VM for agent i at time t.

OGAWR Mechanism runs in each time unit t, and the algorithm is described as follows:

Step 1 According to the allocation rule of OGAWR, the allocation is performed based on S^t, θ^t, which generates an allocation set π^t and a new status set S^{t+1} , and updates allocation result A.

Step 2 For each agent i who got its first unit at step 1, the critical value for agent i at time t is computed using equation $v_{i,t}^c = v_{-i,t}^{s(t)}$. Then, we execute a suppositional allocation in which i is not present, and get suppositional results π_{-i}^t and S_{-i}^{t+1}.

Step 3 For each i who got the allocation before time t and $t \leq d_i - l_i + 1$, according to the suppositional result S_{-i}^t which suppose that i had been not present before t, we compute the critical value for agent i as $v_{i,t}^c = v_{-i,t}^{s_{-i}(t)}$. We also execute a suppositional allocation in which i is not present based on the suppositional status S_{-i}^t, and get suppositional results π_{-i}^t and S_{-i}^{t+1}.

Step 4 For each i who satisfies $t = d_i - l_i$, the payment p_i is computed. If $A_i = 0$, the payment p_i is zero, otherwise the payment can be computed as

$$p_i = (\min_{t \in [a_i, d_i - l_i + 1]} v_{i,t}^c) \cdot l_i$$

Lemma 1 *The payment in above algorithm is a critical payment. That is,* $(\min_{t \in (a_i, d_i - l_i + 1)} v_{i,t}^c) \cdot l_i = V_{(a_i, d_i, V_i, l_i, e_i)}^c(\theta_{-i})$ *for each allocated agent* i.

Table 2. Mechanism algorithm: OGAWR

Input: t, $\theta^t = \{\theta_1, \theta_2, ..., \theta_n\}$, S^t
Output: S^{t+1}, π^t, p_i, A

$\pi^t = \emptyset$
if $s(t) = 0$, goto end
(S^{t+1}, π^t, A)=$Allocate(\theta^t, S^t, t)$
for each $\pi_i \in \pi^t$ do
    $v^c_{i,t} = v^{s(t)}_{-i,t}$
    $(S^{t+1}_{-i}, \pi^t_{-i}, A_{-i})$=$Allocate(\theta^t_{-i}, S^t, t)$
end for
for each $i \notin \pi^t$and $A_i = 1$ and $t \leq d_i - l_i$ do
    $v^c_{i,t} = v^{s_{-i}(t)}_{-i,t}$
    $(S^{t+1}_{-i}, \pi^t_{-i}, A_{-i}) = Allocate(\theta^t_{-i}, S^{t-1}_{-i}, t)$
end for
for each i: $t = d_i - l_i + 1$
    if $A_i = 0$, $p_i = 0$
    else $p_i = \min_{t \in [a_i, d_i - l_i + 1]} v^c_{i,t} \cdot l_i$
end for
end

5 Analysis of OGAWR Mechanism

We assume no early-arrival no late-departure misreports with $a_i \leq a'_i \leq d'_i \leq d_i$, because generally agent i don't know its type until a_i and the value of agent will be zero if it is finished after d_i. We also assume no less job size misreports with $l'_i \geq l_i$ because the agent i will have no sufficient time to process if $l'_i < l_i$.

Definition 4 *(Monotonic with resource demand) An allocation policy is monotonic with resource demand l_i if for any arrival-departure interval $[a_i, d_i]$, any valuation V_i and any job size report $l'_i \geq l_i$, we have $A_i(a_i, d_i, V_i, l'_i) = 1 \Rightarrow A_i(a_i, d_i, V_i, l_i) = 1$.*

Definition 5 *(Monotonic with arrival-departure interval) An allocation policy is monotonic with arrival-departure time if for any job size l_i, any valuation V_i and any arrival-departure time report $a'_i \geq a_i$ and $d'_i \leq d_i$, we have $A_i(a'_i, d'_i, V_i, l_i) = 1 \Rightarrow A_i(a_i, d_i, V_i, l_i) = 1$.*

Lemma 2 *The allocation policy in OGAWR mechanism is monotonic with resource demand and arrival-departure interval.*

Next, we discuss about whether an agent would get more utility by misreport e_i. First, an inflexible agent would not misreport $e_i = 1$ because discontinuous allocation for this class job will cause zero value. Second, we find there is no difference in allocation and payment to flexible agent between reporting $e_i = 1$ and reporting $e_i = 0$. For allocation, due to the greedy allocation and non-increasing reserving, whether an agent can be allocated is only decided by the order of its valuation and not related to e_i. For payment, according to the critical value equation(4), the critical value of agent i would not changes when e_i changes, and

the payment of the agent is equal to the critical value which is also not related to e_i. We assume that each agent is rational, that is, the agent will choose to report true type when misreport cannot improve its utility.

Theorem 1 *The OGAWR mechanism is incentive compatible with no-early arrival, no-late departure misreports and no less job size misreports.*

We define the competitive ratio on social welfare as follows. An auction mechanism M is $c-competitive$ with respect to the social welfare if for every bidding sequence θ , $E_M(\theta) \geq E_{opt}(\theta)/c$. Accordingly, c is the competitive ratio of M. Where, E_M is the sum of agents value in mechanism M, and E_{opt} denotes the sum of agents value by the optimal algorithm.

Assume that VM to all agents has a same maximal unit valuation v_{max} and same minimal unit valuation v_{min}, i.e, $v_i \in [v_{min}, v_{max}]$. Define $N = \frac{v_{max}}{v_{min}}$. At the same time, we assume the maximal job size is L and $L \geq 2$.

Theorem 2 *OGAWR mechanism has a competitive ratio on social welfare* $\frac{C \cdot N \cdot (L+1)}{2}$.

6 Evaluation and Simulation

As analysed above, the competitive ratio c of OGAWR mechanism might be very large because it is decided by L and N. That is, it may lead to very low social welfare at the worst case. In this section, we will present the simulation results and compare the OGAWR mechanism with two allocation methods. one is an offline optimal approach designed under the assumption that we know all the agents valuation beforehand and completely ignore the allocation time constraint in $[a_i, d_i]$. Although it is not reasonable that OGAWR mechanism is compared with the offline allocation without time constraint, we can understand the actual level of proposed mechanism on social welfare and percentage of served agents by comparing the curves. The other method compared is a good online mechanism (MOVMPA) designed in paper [13]. The main difference between MOVMPA mechanism and OGAWR mechanism is that MOVMPA uses continuous allocation for each agent.

Same as [13], the input data of the experiments are collected from the Parallel Workload Archive [15], which collect many workload logs from large scale parallel systems in various places around the world. We select 10 thousands continuous records from log RICC-2010-2. In the log, the minimal time unit recorded is second. In our experiment, we choose 10 minutes as one time unit, and all records selected are distributed randomly from time 0 to time 8000. Each record corresponds to one task, and the information of a task includes arrival time , wait time, runtime, number of allocated processors, etc. Each task is processed by at most 8 thousands processors. According to the number of allocated processors k, a task can be divided into k subtasks each of which must be processed serially in one processor. That is, one subtask requests at most one VM in each time unit which is consistent with the assumption in our model. Let each agent present one subtask(also is one job). After the step of task decomposition, there are about 285 thousands agents in these records.

Next, we discuss how the type of agent $\theta = (a_i, d_i, l_i, e_i, V_i)$ can be got. First, a_i and l_i can be got from the log, where the real arrival time of the record is a_i and the runtime can be converted to the size l_i. As described above, if k agents are generated from one same record, they will have same arrival time and job size. Second, we produce the other information d_i and V_i. Assume that the deadline and the valuation are exponential distribution. Deadline d_i and valuation V_i/l_i are computed as $d_i = a_i + l_i + l_i \cdot exp(d_{avg})$ and $V_i/l_i = exp(v_{avg})$. Finally, the parameter e_i is generated randomly. The table II shows the simulation parameters.

Table 3. Simulation Parameters

Type	Notaion	Value	Parameter
Arrival time	a_i	form workload archive	
departure time	d_i	$a_i + l_i + l_i \cdot exp(d_{avg})$	$d_{avg} = 2$
job size	l_i	from workload archive	
valuation	V_i	$V_i/l_i = exp(v_{avg})$	$v_{avg} = 50$
job character (flexible)	e_i	1 or 0, generate randomly	

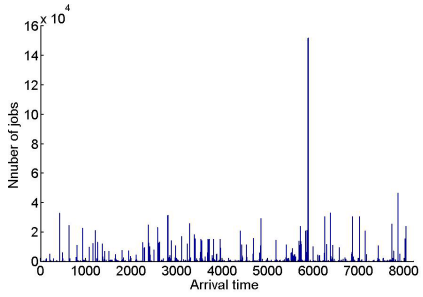

Fig. 2. Distribution of arrival time

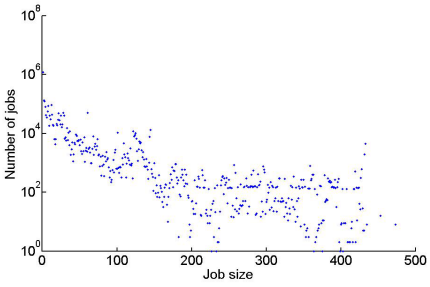

Fig. 3. Distribution of job size

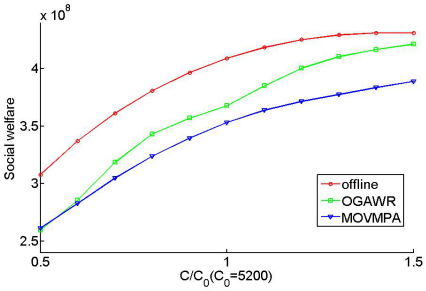

Fig. 4. The number of competed jobs under three mechanisms

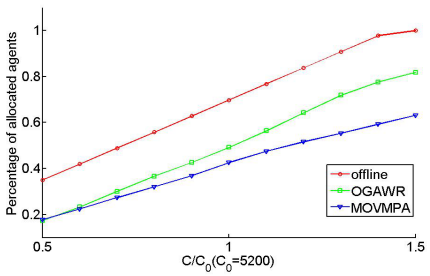

Fig. 5. The sum of agents value under three mechanisms

Fig. 2 and 3 shows the distribution of all those records we selected. Fig. 2 shows the number of arrival subtasks at each time unit, while Fig. 3 is the size distribution of all agents.

Before running of the mechanism, we initialize the supply, the total number of VMs, which is closely related to the allocation results. we define an initial supply C_0 is equal to average requirement for each time unit, i.e., $C_0 = \Sigma_{i=1}^{n} l_i/|T|$, where $|T| = 8000$ is total time units we select.

Fig. 4 shows the social welfare, the sum of agents value with different C, and C changes continuously from $C = 0.5 \cdot C_0$ to $C = 1.5 \cdot C_0$. First, we note that the trends for the two scenarios in OGAWR mechanism are different C when supply is low and close to $C = 0.5 \cdot C_0$, the OGAWR mechanism results only in a small overall improvement in social welfare, However, when it grow to more than C_0, there is a very obvious improvement. Especially, when $C = 1.5 \cdot C_0$, the social welfare in OGAWR mechanism is very close to it in the condition of offline allocation, while it still keep a low level in MOVMPA mechanism.

With respect to the number of completed jobs of individual agents, the results are shown as Fig. 5. The percentage of completed jobs increases with the increase of supply in all allocation approaches, and in OGAWR, it is obviously higher than that in MOVMPA when they are in the same supply capacity.

7 Conclusion

In this paper, we propose an online VM allocation mechanism for private IaaS cloud resources, whose goal is to improve the social welfare. We construct a online resource allocation model in which jobs are divided into two classes: inflexible jobs and flexible jobs. Then, an online greedy allocation with reservation(OGAWR) mechanism under the dynamic cloud environment is designed and proved truthfully. We also performed extensive experiments to observe the results of the mechanism. The results show that, from the aspects of improving social welfare and the number of completed jobs, OGAWR is better than the mechanism which allocates the inflexible agents as well as flexible agents.

For future work we plan to consider several issues. First, in this paper we assumed all agents need only one VM per time unit, but in the future we plan to extend the allocation model to deal with multiple VMs demands per time unit. Second, it would be interesting to design model-based mechanism and compare the performance with the model-free online mechanism proposed in this paper. Finally, we also plan to study online mechanism design for public IaaS cloud in future work, where the goal of mechanism is to maximal the revenue of cloud provider.

Acknowledgments

The workload log from the RICC cluster was graciously provided by Motoyoshi Kurokawa. This Work was supported by the National Natural Science Foundation of China (61170029, 61100052, 61373032, 91318301), Zhejiang Provincial Natural Science Foundation of China under Grant No. Y1111000, and Zhejiang Provincial Science and Technology Plan of China under Grant No. 2013C31097.

References

1. Amazon. Amazon EC2 spot instances. http://aws.amazon.com/ec2/spot-instances/.
2. Nisan, Noam, ed. Algorithmic game theory. Cambridge University Press, 2007.
3. D.C. Parkes and S. Singh. An MDP-Based approach to Online Mechanism Design. In Proc. of NIPS03, 2003.
4. A. Gershkov and B. Moldovanu. Efficient sequential assignment with incomplete information. Games and Economic Behavior, 68(1):144-154, 2010.
5. E.H. Gerding, V. Robu, S. Stein, et al. Online mechanism design for electric vehicle charging. In Proceeding of the 10th International Conference on Autonomous Agents and Multiagent Systems(AAMAS 2011), 811-818, 2011.
6. A. Nahir, A. Orda, and D. Raz. Workload Factoring with the Cloud: A Game-Theoretic Perspective. In Proceedings of the 31st Annual Joint Conference of the IEEE Computer and Communications Societies. Networking (INFOCOM'12), IEEE Society, 2566-2570, 2012.
7. G. Wei, A.V. Vasilakos, Y. Zheng, et al. A game-theoretic method of fair resource allocation for cloud computing services. The Journal of Supercomputing, 2010, 54(2): 252-269.
8. S. Zaman, D. Grosu Combinatorial auction-based allocation of virtual machine instances in clouds. Journal of Parallel and Distributed Computing, 2013, 73(4): 495-508.
9. Q. Zhang, E. G urses, R. Boutaba Dynamic resource allocation for spot markets in cloud computing environments. 2011 Fourth IEEE International Conference on Utility and Cloud Computing (UCC). IEEE, 178-185, 2011.
10. A. Danak and S. Mannor. Resource allocation with supply adjustment in distributed computing systems. In proceeding of the 30th International Conference on Distributed Computing Systems (ICDCS). IEEE, 498-506, 2010.
11. Q. Wang, K. Ren, and X. Meng. When cloud meets eBay: Towards effective pricing for cloud computing. In Proceedings of the 31st Annual Joint Conference of the IEEE Computer and Communications Societies. Networking (INFOCOM'12), IEEE Society, 936-944, 2012.
12. N. Jain, I. Menache, J. S. Naor, et al. A truthful mechanism for value-based scheduling in cloud computing. Algorithmic Game Theory. Springer Berlin Heidelberg, 2011: 178-189.
13. S. Zaman, D. Grosu. An Online Mechanism for Dynamic VM Provisioning and Allocation in Clouds. In proceeding of the 5th International Conference on Cloud Computing (CLOUD). IEEE, 253-260, 2012.
14. H. Zhang, B. Li, H. Jiang, et al. A framework for truthful online auctions in cloud computing with heterogeneous user demands. In Proceedings of the 32st Annual Joint Conference of the IEEE Computer and Communications Societies. Networking (INFOCOM'12), IEEE Society, 1510-1518, 2012.
15. D. G. Feitelson. Parallel Workloads Archives: Logs. http://www.cs.huji.ac.il/labs/parallel/workload/logs.html.
16. X. wu, Y. Gu, G. Li, et al. Online Mechanism Design for VMs allocation in Private Cloud. http://basics.sjtu.edu.cn/ liguoqiang/paper/Onlinefull.pdf.