A Game Theoretical method for Auto-Scaling of Multi-tiers Web Applications in Cloud

Ruiqing Chi
Department of Computer Science and Technology
Nanjing University
Nanjing Jiangsu, China
chiruiqing@dislab.nju.edu.cn

Zhuzhong Qian
Department of Computer Science and Technology
Nanjing University
Nanjing Jiangsu, China
qzz@nju.edu.cn

Sanglu Lu
Department of Computer Science and Technology
Nanjing University
Nanjing Jiangsu, China
sanglu@nju.edu.cn

ABSTRACT

Cloud computing is a newly emerging reliable and scalable paradigm in which customers pay for cloud resources they use on demand. However, current auto-scaling mechanisms in cloud lack the critical self-adaption policy which helps application providers decide on when and how to reallocate resources. Furthermore, virtualization techniques can not ensure an absolute isolation between multiple virtual machines sharing the same physical resource, which leads to some customers paying unfairly for heavy-loaded resource under a widely-adopted fixed pricing scheme.

In this paper, we present a global performance-to-price model based on game theory, in which each application is considered as a selfish player attempting to guarantee QoS requirements and simultaneously minimize the resource cost. Then we apply the idea of Nash equilibrium to obtain the appropriate allocation, and an approximated solution is proposed to obtain the Nash equilibrium, ensuring that each player is charged fairly for their desired performance. First, each player maximizes its utility independently without considering the placement of virtual machines. Then based on the initial allocation, each player reaches its optimal placement solely without considering others' interference. Finally we propose an evolutionary algorithm which step by step updates the global resource allocation based on the initial optimal allocation and placement.

Categories and Subject Descriptors

C.4 [**PERFORMANCE OF SYSTEMS**]: Design studies; C.2.4 [**COMPUTER-COMMUNICATION NETWORKS**]: Distributed Systems—*Distributed Applications*

General Terms

Algorithms, Experimentation, Performance

Keywords

Game theory; Cloud computing; Resource allocation; Auto scaling; QoS; Web application.

1. INTRODUCTION

Cloud computing [4], which implies that computing is on-demand processed on both local computer and third-party distributed facilities, is gaining a lot of momentum in both industrial and academic circles. There is an explosion of both public and private clouds in recent years, such as Google AppEngine [2], Amazon Web Services(AWS) [1] and Microsoft Azure [3]. Thus aiming to undercut their expense, more and more application providers tend to deploy their web applications in cloud and rent resources in a pay-as-you-go manner. The workload of web applications has not only periodical variations such as time-of-day effects, but also short-terms crowds due to some web hot spots. As a result, it is an important issue for cloud providers to decide how to allocate resources dynamically to satisfy QoS requirements from multiple clients as workload varies widely and unpredictably. However, there are several intricacies in this problem.

Firstly, current auto-scaling mechanisms are quite preliminary. For example, Amazon Auto Scaling [5] is designed to launch or terminate EC2 instances automatically when some performance metrics(e.g. CPU utilization) of virtual machines(abbreviated to VM) go above or fall below the presetting threshold. It only considers the runtime performance of virtual machines, ignoring the QoS requirements of web applications running on these virtual machines. Actually, clients care more about whether their QoS requirements are satisfied.

Secondly, modern web applications generally apply a complex multi-tiers architecture, which is composed of dozens of subsystems. For instance, a typical three-tiers e-commerce application consists of a front-end web server, an application logic tier and a back-end database server. In each tier, there are several virtual instance replicas and workload is dispensed averagely to each replica by workload balancer. Further, they also have drastically different kinds of transactions, which have widely variable executing paths and resource demands. Hence, it is not so cushy to design an efficient and accurate resource allocation policy for determining which is the bottleneck tier and how much resource to reallocate in each tier.

Finally, current virtualization technology cannot guarantee a complete isolation between virtual machines sharing the same physical resource. This is usually due to CPU scheduling and I/O sharing among multiple virtual machines on the same physical machine. As a result, there exists a complex interplay between clients sharing the cloud resources. However, current existing fixed pricing schemes don't consider this complex client-client interaction, and thus it's not quite fair for all clients. For example, a request processing takes longer time when the same VM is located on a heavily loaded physical machine in contrast to a lightly loaded one. Meanwhile, current fixed pricing mechanism, such as a per cpu/hour cost scheme used by Amazon, also implies that clients have to pay more for processing the same amount of requests on high-loaded physical resource when the configuration of VM is unchangeable. Obviously, it's really unfair and unacceptable for clients who rent high-loaded resources.

In this paper, we propose a novel QoS constrained cloud resource allocation mechanism which captures the inherent behaviors of multi-tiers web application and quantifies the degradation in application performance due to client-client interference. Particularly, game theory is used to model the inherent tradeoff between application performance and resource cost. Thus we model the cloud resource allocation problem as a new class of non-cooperative game [15] in which each client selfishly tries to maximize its own utility. In other words, each client is aimed to satisfy its QoS requirements and simultaneously minimize its resource cost. Therefore it's proper to define the utility function of a client as a weighted sum of the performance benefit and the expense of the leased resource. We model the performance benefit as a measure of how the QoS requirements of web application are satisfied. And considering the client-client interaction, we define the resource price function as a monotonic non-increasing function of resource utilization. Besides, we use an LQN(Layered Queuing Network) model [13] [11] to predict the application performance behaviors precisely over a a wide range of workload and resource allocation schemes.

Then we apply the idea of Nash equilibrium to determine the appropriate resource allocation for all clients. In a Nash equilibrium, nothing will incentivize a client to unilaterally change its resource allocation strategy. Unfortunately, due to the tremendous parameter space, it's PPAD-complete [9] to find the optimal resource allocation. To obtain the Nash equilibrium in an effective and efficient way, an approximated solution is proposed by solving the following three relevant problems. i)IRAP(independent resource allocation problem): each player obtains its optimal allocation independently without considering the placement of virtual machines. ii)IVMPP(independent virtual machine placement problem): each client solely maps its virtual machines to all available physical resource without considering client-client interaction. iii)MRAPP(multiplexed resource allocation and placement problem): based on the initial allocation and placement, an evolutionary algorithm is designed to update the global initial resource allocation step by step until no one can increase its utility by unilaterally changing its strategy.

Our paper makes four main contributions: i) propose a local search algorithm for the optimization of IRAP; ii) propose an evolutionary algorithm for the optimization of IVMPP based on the initial resource allocation obtained from IRAP; iii) design a game theoretic resource allocation mechanism which can reach a fair resource allocation state for MRAPP; iv) demonstrate that the resource allocation result of our algorithm is actually a Nash equilibrium.

The rest of this paper is organized as follows. We examine our related work in section 2. Then the next section provides an overview of global resource allocation problem. Section 5, 6 and 7 discuss our overall solution in detail. In section 8, we present our experiment and verify the effectiveness of our approach. Finally, we draw our conclusions and discuss our future work in section 9.

2. RELATED WORK

Cloud computing is a scalable computing paradigm in which consumers rent and release the resource dynamically in order to meet their QoS requirements with minimal resource cost. Therefore, an efficient and effective global resource allocation strategy is very important for cloud providers.

As we have investigated, many techniques have been proposed for the problem of dynamic resource allocation in data center, such as control theory([6]) and stochastic model([8], [19]). They all follow the same paradigm: i) build the application performance model and then fix some model parameters by measurement or machine learning; ii) design a scheduling strategy to maximize the utility function [13]. [24] uses a table-driven approach which stores response time values for different workloads and resource allocation schemes. However, it doesn't scales well with multiple resource types and transaction classes. Queuing network model is used in [6], [7]and [10], but only for single tier server systems, while multi-tiers applications are all considered in [13] [20] [21] [22]. [17] further proposes an cooperative elastic resource provision mechanism between the private infrastructure and public cloud. [18] considers the nonlinear pricing scheme and proposes an online control framework.

However, all these work are not aware of the inherent client-client interference in cloud computing environments. Economic approaches have been successfully adopted to capture the complex interactions between clients of a shared computing system in [14] and [16]. [12] has proposed a game theory model but don't take into account the interference between the clients and the performance seen by them, while [23] considers these important properties of real cloud environments and make some theory analysis on the Price of Anarchy of Nash equilibrium. The work closest to ours is [25], where game theory is used to solved the dependent task scheduling problem, but the author assumes that each resource is totally occupied and shared by several tasks. In this paper, we use game theory to solve the global resource allocation and VM placement problem with considering the client-client interference.

3. PROBLEM MODEL

Suppose that cloud provides a set of m physical machines $M = \{1, 2, \ldots, m\}$ to multiple applications. Each machine i has a capacity c_i which represents the total amount of available CPU resource at machine i. For modeling the interference between multiple clients sharing the physical re-

source, we introduce a dynamic resource pricing function $p_i(x)(0 \leq x \leq c_i)$. x is the amount of already occupied resource in machine i. And $p_i(x)$ represents the unit price of machine i on which x unit resource has already been used. The pricing function p is an abstract cost that clients pay for completing the same amount of jobs on machines with different resource utilizations. Generally, the higher resource utilization is, the more time clients need to finish a job, the more money clients have to pay. Put differently, the pricing function is directly proportional to the resource utilization and the time taken to complete a job. Thus it is proper to define a monotonic non-decreasing function $p_i(x)$ as:

$$p_i(x) = ax + b(0 \leq x \leq c_i) \tag{1}$$

Here a, b is constant. The price p increases as the resource utilization x increases.

Then we further assume that a set $A = A_1, A_2, \ldots, A_n$ of n multi-tiers web applications sharing the cloud resource M. Let k_i be the amount of the tiers in application A_i. And for each tier $j(1 \leq j \leq k_i)$ of application A_i, let C_{ij} be the amount of virtual instance replicas. Without loss of generality, we assume that each replica at the same tier has an equal capacity x_{ij}.

Each application provides a variety of transactions and then assume that a set $T_i = t_{i1}, t_{i2}, \ldots, t_{il_i}$ represents the transactions of application A_i. l_i stands for the number of transactions. For each transaction t_{ij}, the workload intensity is characterized by its request rate w_{ij}. Here we mainly consider the response time as the relevant performance metric of each transaction. Thus each transaction t_{ij} is associated with an actual average response time r_{ij} and a target response time r_{ij}^T which denotes the QoS requirements of transaction t_{ij}. The response time can be computed by a performance model LQN presented in next section.

A global resource allocation vector $\Phi = (\phi_1, \phi_2, \ldots, \phi_n)$ defines the amount of virtual instance replicas allocated by cloud to each application at all machines. ϕ_i, which represents the strategy adopted by application A_i, is given by a non-negative matrix of k_i rows,one for each tier, and m columns, one for each physical machine. The entry $\phi_{ij}(s)$ indicates the amount of the jth tier replica of application A_i allocated to machine s. The global allocation Φ is feasible if it satisfy the following conditions:(1) $\phi_{ij}(s) > 0, \forall i, j, s$; (2) the total occupied resource R_s at machine s cannot exceed its capacity c_s, i.e. $R_s = \sum_{i=1}^{n} \sum_{j=1}^{k_i} \phi_{ij}(s) \leq c_s$.

For the users of application A_i, they only care about whether QoS requirements are satisfied. Thus we define the performance benefit of application A_i as the degree of user satisfaction which is given by

$$Benefit_i(\Phi) = \sum_{j=1}^{k_i} 1 - \frac{1}{1 + e^{r_{ij}^T - r_{ij}}} \tag{2}$$

In fact, [24] has discussed the selection of utility function in autonomic computing and shown that other functions also can be viable as long as they are monotonically non-increasing with increasing response time. However, our benefit function, which is a very intuitive reflection of user satisfaction, has an inflection point at r_{ij}^T and changes slowly when response time is away from the target response time. See a example in Fig.1 for $r_{ij}^T = 3.5$.

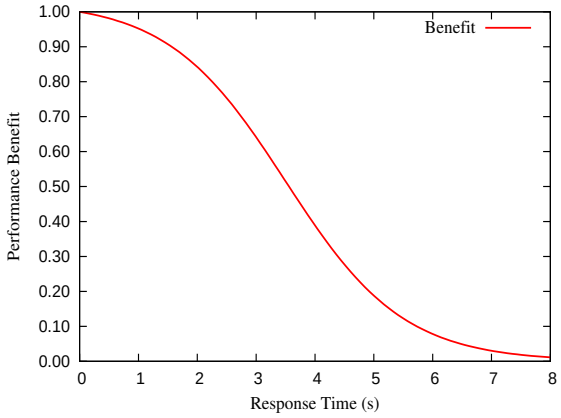

Figure 1: Performance Benefit Function($r_{ij}^T = 3.5$)

In general, there exists an inherent tradeoff between performance benefit and resource cost for each application. Assume that each application provider has the same view of value towards performance and cost, we define the utility of each application as a sum of the resource cost and performance benefit. Therefore, each application provider wishes to maximize its utility which is given by

$$U_i(\Phi) = Benefit_i(\phi) - \sum_{s=1}^{m} \sum_{j=1}^{k_i} \phi_{ij}(s) x_{ij} p_s(R_s) \tag{3}$$

Let each application be a market participant, the 4-tuple $\{A, M, \Phi, U\}$ defines a cloud resource allocation and placement game where each player attempts to solve the following optimization problem:

$$Maximize \quad U_i(\Phi) \tag{4}$$

$$Subject\ to \quad \sum_{i=1}^{n} \sum_{j=1}^{k_i} \phi_{ij}(s) x_{ij} \leq c_s \tag{5}$$

$$\phi_{ij}(s) > 0, \forall i, j, s \tag{6}$$

In this game, there exists an inherent conflict of interest between multiple clients: each client selfishly tries to increase its utility by modifying its strategy ϕ_i until the system reach a Nash Equilibrium [15] state in which none can increase its utility unilaterally. Assume that each client can change its allocation strategy only by the following three one-step actions:

$$add(i, j, s) = \phi_{ij}(s) + +$$
$$reduce(i, j, s) = \phi_{ij}(s) - -$$
$$migrate(i, j, s, q) = \phi_{ij}(q) + +; \phi_{ij}(s) - -$$

$add(i, j, s)$ increase the amount of VM ϕ_{ij} on machine s by one, while $reduce(i, j, s)$ decrease the amount by one. $migrate(i, j, s, q)$ represents the migration of VM ϕ_{ij} from machine s to q. Thus, each strategy can be transformed from an arbitrary initial allocation by a series of actions

above. In a Nash equilibrium, cloud providers guarantee a fair resource allocation for all the clients. According to the general definition of Nash equilibrium, we define a *Nash Allocation* for our game as follows:

***Definition** 1.* A feasible resource allocation Φ is said to be a *Nash Allocation* if $\forall A_i \in A$, $U_i(\phi_i^{'}, \phi_{-i}) \leq U_i(\phi_i, \phi_{-i})$, for any feasible allocation $\phi_i^{'}$ which satisfies the capacity constrained condition. According to the game theory convention, ϕ_{-i} is used to denote the remaining elements of Φ when ϕ_i is dropped out of Φ.

Unfortunately, it is PPAD-complete [9] to find a *Nash Allocation* for the fair resource allocation problem. To obtain the Nash allocation in an effective and efficient way, we propose an solution by solving the following three relevant problems:

1. IRAP(Independent Resource Allocation Problem): each player tries to reach its optimal allocation independently without considering the placement of virtual machines (Sect.5).

2. IVMPP(Independent Virtual Machine Placement Problem): based on the optimal allocation, each player attempts to maximize its utility by mapping the allocated VMs to the physical resource independently and solely with considering the dynamic pricing(Sect.6).

3. MRAPP(Multiplexed Resource Allocation and Placement Problem): based on the optimal allocation and placement of each player, an evolutionary algorithm is designed to update the global initial optimal resource allocation step by step until no one can increase its utility by unilaterally changing its strategy(Sect.7).

4. APPLICATION PERFORMANCE MODEL

For estimating the response time of multi-tiers web application, we apply a layered queuing network model to predict the performance behavior accurately over a variety of workload and resource allocation. In LQN model, each hardware resource and software server of multi-tiers application is modeled as a task, which has one or more service entries representing different operations it can execute. For example, a web server may offer more than one transaction with different service time. Each task is modeled by using an $M/M/n$ queue where n represents the concurrence of the hardware or software components. Each task can make requests and all requests need queuing at the entry of other tasks. Once a request is accepted, it then executes a service there and sometimes maybe invoke any number of nested service at other tasks. Thus we can capture the multi-tiers feature effectively by using LQN model whose essence is to capture the nesting of requests. And this model is also well scalable with multiple resource types and enable a request to consume multiple resources simultaneously.

Assume that the average service time and workload of transaction t_{ij} at tier λ is respectively s_{ij}^{λ} and w_{ij}^{λ},we can solve the layered queuing network model through mean-value analysis algorithms to get the end-to-end response time which is given by:

$$r_{ij} = \sum_{\lambda=1}^{k_i} \frac{s_{ij}^{\lambda}}{1 - \sum_{\mu=1}^{l_i} \frac{w_{i\mu}^{\lambda}}{\phi_{i\lambda}} s_{i\mu}^{\lambda}} \tag{7}$$

The layered queuing network model is used to predict the performance metrics of multi-tiers cloud applications by our global resource allocation mechanism discussed in the sections below.

5. IRAP OPTIMIZATION

In this section, we discuss the optimization of the independent resource allocation problem(IRAP) which concerns only one application solely. Each application tries to reach its optimal allocation independently and solely without considering both the client-client interaction and dynamic pricing. In other words, the unit price of each resource is fixed and let the unit price be β. And the actual placement of virtual machines is not accounted for in IRAP. Thus we can transfer the original problem into the following optimization problem with different constraints:

$$Maximize \qquad Benefit_i(C_i) - \sum_{j=1}^{k_i} C_{ij} x_{ij} \beta \tag{8}$$

$$Subject\ to \qquad C_{ij} > 0, \forall i, j \tag{9}$$

Here, the vector $C_i = (C_{i1}, C_{i2}, \ldots, C_{ik_i})$ stands for the resource allocated by cloud to the application A_i, where each C_{ij} denotes the amount of the virtual instances at the jth tier. And Equation (8) denotes the utility of each application in IRAP, where the resource price does not change with resource utilization.

To tackle the optimization problem in a really efficient way, we make use of some interesting pieces of information on the running states of web application. On inspection we find that the performance of web application can not be worse when it is configured with more servers. In other words, the more servers, the better performance. So we can traverse over all the parameter space by add and reduce action. In each step, we only execute one add or reduce action on current configuration to get its neighbor node which is defined as:

***Definition** 2.* Assume that $C_1 = (C_{11}, C_{12}, \ldots, C_{1M})$ and $C_2 = (C_{21}, C_{22}, \ldots, C_{2M})$, C_1 is the neighbor node of C_2 if and only if there is one and only one $t(0 \leq t \leq M)$ such that $C_{1t} = C_{2t} \pm 1$.

In this section, we propose a local allocation optimization algorithm to get the optimal configuration scheme of IRAP. Start with an initial configuration, we traverse its all neighbor nodes and keep some branches with the higher utility for the next cycle. Thus we can continue our search in the right direction of optimal configuration until no neighbor node is able to increase the utility. Our algorithm is demonstrated as below.

Thereinto, Ω denotes the amount of the candidate configurations with the highest utility values, which we keep for

Algorithm 1 Local Allocation Optimization

Input:
W_{ini}-the initial workload
C_{ini}-the initial configuration
Output:
C_{opt}-the optimal configuration

```
ConfList ⟵ C_ini
U_max = U_ini; flag = true
while flag do
  for all neighbor nodes of C in ConfList do
    compute U for all nodes
    ConfList ⟵ Ω max configuration
  end for
  if U_max > U for all in ConfList then
    flag=false;
  else
    U_max ⟵ MaxUtility(ConfList)
    C_opt ⟵ Max(ConfList)
  end if
end while
```

the next cycle. Then we repeat the search process until an optimal configuration is found.

***Proposition** 1.* The result of algorithm 1 must be the optimal allocation of independent resource allocation problem.

PROOF. Let C_i be the final result of algorithm 1. If C_i is not optimal, there must exist a different resource configuration C_{opt}, which makes $U(C_{opt}) > U(C_i)$. Therefore, there is a transformation path between two configurations through a series of add and reduce actions. Each action in the path from C_i to C_{opt} must increase the utility. Otherwise the utility of the configuration without those actions which decrease the utility must be higher than C_{opt}. This is against to our assumption that C_{opt} is optimal. Thus it's reasonable to consider C_{opt} is the neighbor node of C_i with a higher utility. However, it's also contrary to the fact that our algorithm won't stop until the utility of all neighbor node is lower than C_i. Therefore, C_i must be the optimal resource allocation. □

Using the above technique, we are able to get an optimal configuration with modest computation. Obviously, the performance of our algorithm is closely associated with the selection of an initial configuration. To avoid some unnecessary search, we can store some valuable configuration-workload couples in database. When our algorithm is invoked, we can compute the Euclid distance between current workload and each workload in database, and then take the couple with the nearest distance as the initial configuration.

6. IVMPP OPTIMIZATION

As described in sect.5, each application has got their independent optimal allocation without considering either the client-client interaction or dynamic pricing. In this section, given the optimal resource configuration C_i of each application, we try to solve the the independent virtual machine placement Problem(IVMPP), i.e. the maximization of its utility by mapping the virtual machines to the physical machines solely without considering the client-client interaction. Even if there is only one application in cloud, the virtual machines sharing the same resource will still interfere with each other. We apply a dynamic pricing mechanism to model the inherent performance interference. Therefore, it's natural to skip the unchangeable performance benefit part in utility function and define IVMPP as:

$$Maximize \quad -\sum_{s=1}^{m}\sum_{j=1}^{k_i}\phi_{ij}(s)x_{ij}p_s(R_s) \tag{10}$$

$$Subject\ to \quad \sum_{i=1}^{n}\sum_{j=1}^{k_i}\phi_{ij}(s)x_{ij} \le c_s \tag{11}$$

$$\phi_{ij}(s) > 0, \forall i,j,s \tag{12}$$

$$\sum_{s=1}^{m}\phi_{ij}(s) = C_{ij}, \forall j \tag{13}$$

Inequation (11) denotes that the occupied capacity in each machine is within its overall capacity. And Equation (13) means the constraint of virtual machine allocation.

Based on the optimal resource configuration in IRAP, we firstly generate the initial virtual machine assignment through a brief and intuitive placement. In each step, we put only a virtual machine in the physical machine with the least $\Delta costgain$ which is given by

$$\Delta costgain_s = (R_s + x)p(R_s + x) - (R_s)p(R_s) \tag{14}$$

After the iteration procedure, an initial placement is generated, but it may be not the optimal placement. Based on the initial resource allocation obtained in IRAP, we can reach the optimal placement by migration. To reach an optimal placement, we propose a step-by-step adjustment algorithm in which each adjustment must satisfy the following condition: a migrate action can be executed if and only if it will increase the utility. And assume that one and only one action can be executed in each step of the adjustment process. However, there are multiple possible destinations that a virtual machine can be migrated to. For characterizing the utility loss of different migration, we define a variable VMMLG(VM Migrate Local Gain) as

***Definition** 3.* VMMLG is the amount by which the utility of the prior placement is greater than the later one, when a virtual machine is migrated from one machine to another.

So we need to find such a machine with the minimum value of VMMLG when performing a migrate action. This can be easily solved and we define the process as a function $minPos(i,j,s)$. This function returns the target machine where the jth tier virtual machine at machine s should be migrated. If there does not exist a replacement of this virtual machine which increase the utility, the function returns 0.

Besides, there may be multiple virtual machines which belong to different tiers of the application at the same machine. So it is possible that there are several types of virtual machines which can be migrated to decrease the utility. Thus,

Algorithm 2 minVM(i,s)

Input:
i-the application
s-the physical machine
Output:
q-the target tier
construct the VMSet of A_i in machine s
$q = 0$;VMMLG$_{max} = 0$
for all $j \in$ VMSet **do**
  if $minPos(i, j, s) \neq 0$ **then**
    compute its VMMLG
    if VMMLG>VMMLG$_{max}$ **then**
      $q = j$;VMMLG$_{max}$ =VMMLG
    end if
  end if
end for

we define a function *minVM(i,s)* to obtain the virtual machine with the minimum VMMLG. The computing process is shown in algorithm 2.

Based on algorithm 2, the local placement optimization (as described in algorithm 3) is designed to generate the optimal placement for each application independently. The algorithm executes an iteration process of replacement to increase the utility. If there is more than one virtual machines with negative VMMLG in each cycle, we choose the one with the minimum value of VMMLG to migrate in order to fairness.

Algorithm 3 Local Placement Optimization

Input:
ϕ_{ini}-the initial placement
C_{ini}-the initial configuration
Output:
ϕ_{opt}-the optimal placement
construct initial placement ϕ_{ini} by C_{ini}
flag=true
while flag **do**
  for all $s = 1 : m$ **do**
    $vm = minVM(i, s)$
    if $vm \neq 0$ **then**
      $migrate(i, vm, s, minPos(i, vm, s))$
      $flag = false$
    end if
  end for
  $flag =!flag$
end while

7. MRAPP OPTIMIZATION

This section discusses the original multiplexed resource allocation and placement problem(MRAPP). In this game, each player doesn't know the global resource allocation until every one firms up its strategy. As described above, each client has chosen its optimal allocation and placement strategy independently without considering client-client interaction. Thus, the global resource allocation strategy, which is comprised of the independent optimization strategy of each player, have three potential cases:

1. some resource is non-multiplexed by multiple applications. In this case, the application in such resource cannot increase its utility by unilaterally migrating the virtual machines from these resource to others.

2. some resource is used out of its capacity. In this special case, the virtual machines on these resource must be partially reduced or migrated to other resource for meeting the capacity constraint. Since cloud generally provides a huge resource pool, we don't focus on the resource feasibility, i.e. the physical resource is sufficient to accommodate all the virtual machines.

3. some resource is shared by multiple applications without exceeding its capacity. On this condition, each application sharing these resource will attempt to increase its utility by actively change their strategy.

Generally, there may be more than one candidate applications which all be able to increase its utility by adjusting its strategy. However, only one application can adjust its strategy at one time. In this special case, in order to determine fairly which one should be readjusted firstly, we define a variable VMMGG(VM Migrate Global Gain) which denotes the change of the global utility before and after the action. Here, the global utility means the sum of the utility of all application. In the same way we define $minVM(i, j, s)$, $minApp(\Phi, s)$ in algorithm 4 decide which application should be migrated.

Algorithm 4 $minApp(\Phi, s)$

Input:
Φ-the global initial placement
s-the physical machine
Output:
q-the target application
construct the AppSet in machine s
$q = 0$;MaxAppSet= *null*
for all $i \in$ AppSet **do**
  $j = minVM(i, s)$
  if $j \neq 0$ **then**
    put i in MaxAppSet
  end if
end for
if MaxAppSet$\neq$ *null* **then**
  $q \longleftarrow$ minVMMGG(MaxAppSet)
else
  $q = 0$
end if

Assume that each player follows the same evolutionary mechanism and have a full knowledge of cloud. Based on Alg.4, we present a global update mechanism in Alg.5 to obtain the *Nash Allocation*. Each player will actively change its allocation and placement strategy as long as they can increase its own utility. And at one time, there is one and only one player who can execute one-step adjustment action to change its strategy. Beside migration, the player also can increase its utility by a reduce action. This is because some VMs on these highly utilized machine have poor performance and make few contributions on the promotion of global utility. As to the add action, the performance benefit

increase is totally offset by the increase of the resource cost.

Algorithm 5 Global Allocation and Placement Optimization

Input:
ϕ_{ini}-the initial placement
Output:
ϕ_{opt}-the optimal placement

```
revise the initial placement φ_ini
reduceFlag = true
while reduceFlag do
  migrateFlag = true
  while migrateFlag do
    construct the multiplexed resource set resSet
    for all s ∈ resSet do
      app = minApp(Φ, s)
      if app ≠ 0 then
        tier = minVM(app, s)
        migrate(app, tier, s, minPos(app, tier, s)
        migrateFlag = false
      end if
    end for
    migrateFlag =!migrateFlag
  end while
  for all φ ∈ Φ do
    compute its VMMLG for reduce() action
  end for
  vmSet = {vm|VMMLG < 0}
  if vmSet ≠ null then
    make reduce action for the minimum in vmSet
    reduceFlag = false
  end if
  reduceFlag =!reduceFlag
end while
```

The update algorithm is mainly divided into two parts: migration and reduction. At each cycle of migrating process, we select the appropriate virtual machine, which incurs the minimum value of $VMMLG$ and $VMMGG$ simultaneously, and then perform the corresponding migrate action. We will repeat the migration procedure until no player can increase its utility by migrating its virtual machine unilaterally. Then we get into the reduction process in which VMMLG is likewise computed for each node of the global resource allocation Φ. Finally, we choose the one with minimum VMMLG and execute the reallocation by the corresponding reduce action. The whole cycle won't terminate until no one can increase its utility solely by any strategy change.

***Proposition** 2.* The Global allocation obtained by the update algorithm forms a *Nash Allocation* of the resource allocation and placement game.

PROOF. Without loss of generality, let ϕ_i be the optimal allocation strategy of any arbitrary application A_i, and ϕ_{-i} be the corresponding strategies of others. If this global allocation is not a Nash equilibrium, there must be a feasible strategy $\phi_i^{'}$ which satisfy the following condition: $U_i(\phi_i^{'}, \phi_{-i}) > U_i(\phi_i, \phi_{-i})$. Here we prove that such a $\phi_i^{'}$ doesn't exist.

According to our model, a better allocation strategy could be transformed from a worse strategy by a series of actions. Therefore, it's quite reasonable to take $\phi_i^{'}$ as an one-step optimization result of ϕ_i. Put differently, $\phi_i^{'}$ can be reached from ϕ_i by taking one of three available actions: add,reduce and migrate. Thus, there are three different cases:

1. Add action
   In this case, performance benefit will increase and simultaneously resource cost also increases. However, according to the discussion of IRAP, the performance gain is totally offset by the cost gain on fixed-price condition, which is less than the cost gain on dynamic-price condition. Therefore, an add action which increases its utility doesn't exist.

2. Reduce action
   In this case, there is only one different assignment between $\phi_i^{'}$ and ϕ_i. Assume that $\phi_{ij}^{'}(s) = \phi_{ij}(s) - 1$ and j is unique. According to the prior assumption, the utility of $\phi_i^{'}$ is greater than ϕ_i by one reduce action, i.e. $U_i(\phi_i^{'}, \phi_{-i}) > U_i(\phi_i, \phi_{-i})$. However, this is also the only premise on which an feasible action is taken in our algorithm. It is thus inferred that the algorithm won't be terminated if there exists one j and s which makes $U_i(\phi_i^{'}, \phi_{-i}) > U_i(\phi_i, \phi_{-i}) \wedge \phi_{ij}^{'}(s) = \phi_{ij}(s) - 1$. This also implies that ϕ_i can not be the final result of our algorithm. It's against the basic assumption.

3. Migration
   This case can be proved in the same way of (2).

□

8. EXPERIMENT EVALUATION

In this section, we present experimental results to demonstrate the effectiveness of our overall algorithm. To start with, we verify how the LQN model performs in practice in the next subsection.

8.1 LQN Model Accuracy Validation

In our approach, we must ensure that LQN model, which forms the basis of our solution, can predict the response time over a broad range of workloads and configuration schemes. For verifying the accuracy of LQN model, we deploy a book web site in a small private cloud which we built with xen virtualization environment and eucalyptus private cloud. The cloud is mainly made up of three components: cloud controller, cluster controller and node controller. Then we run the book site with a web server and a database respectively in virtual instances. Thus we can scale the site by adding or reducing virtual instances.

Furthermore, we pick two representative transactions, which are respectively "sort" and "browse", to measure their realistic performance. "Browse" is a lightweight transaction and not involved with database while "sort" is heavyweight executes heavy data processing jobs. And we use Httperf tool to generate workload which mixes any kinds of transactions. The measurement and prediction of the response time under the configuration scheme $(1, 1)$ is illustrated in Fig.2.

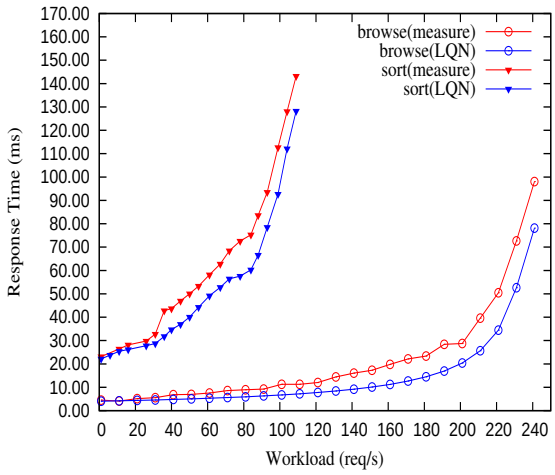

Figure 2: Response Time of browse and sort by measurement and LQN prediction

Fig.2 demonstrates the response time of "browse" and "sort", which is respectively obtained by both actual measurement and LQN model prediction. From the diagram, we can see that the error between the measurement and prediction is less than $20ms$ for "browse" and "sort". Besides, we don't show the response time of "sort" when its workload exceeds $125req/s$. That is because the resource gets saturated and cannot handle more requests. As additional requests are coming one by one, many requests are not responded and have a long response time. Overall, our experiment shows that LQN model corresponds well with the prototype application.

8.2 Algorithm Performance Evaluation

In this subsection, we design a use case to demonstrate the effectiveness of our algorithm as the following. (1)there are three 2-tiers applications(A_1, A_2, A_3), each of which is an independent book site. The current workload of A_1, A_2 and A_3 is respectively (100,100),(200,200)and(300,300).(2)there are five physical machines with the same pricing function $p(x) = 0.5x + 1$.

Thus the overall algorithm is run to get the global optimal allocation and placement which is demonstrated as below.

Step 1: IRAP Optimization

For each application, we run algorithm 1 to get its independent optimal allocation. To application A_1, the utility is computed for the neighbor node $(2, 1)$ and $(1, 2)$ of current configuration $C_{ini} = (1, 1)$. And $(1, 2)$ is found with the maximum utility and kept for the next cycle. By this iteration process, the optimal allocation $(2, 3)$ are obtained for application A_1. In the same way, we can respectively get the optimal allocation $(4, 6)$ and $(6, 8)$ for application A_2 and A_3.

Step 2: IVMPP Optimization

Based on the initial allocation obtained at step 1, algorithm 3 is run to map the allocated VMs to the physical machines for maximizing the utility. To application A_1, we first construct the initial placement and then attempt to maximize the utility by migrating the virtual machines. Finally, the optimal placement matrix obtained is

$$\phi_1 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

Likewise, we can get the independent optimal placement respectively for application A_2 and A_3 as

$$\phi_2 = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 2 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$\phi_3 = \begin{pmatrix} 1 & 1 & 1 & 1 & 2 \\ 2 & 1 & 2 & 2 & 1 \end{pmatrix}$$

Step 3: MRAPP Optimization

In this step, we first compute the application which can be migrated to other resources. To resource m_1, no migration can be found to increase the utility after computing the VMMLG of all possible migration of virtual machines located at resource m_1. In the same way, the VMMLG of any migration can be computed for other resources $m_2 - m_5$. Finally, we find that there does not exist a migration which can increase the utility. Thus the migration process is terminated and then we go into the reducing process.

To application A_1, no reduce action can be executed to increase the utility, i.e. ϕ_1 is the optimal strategy on the current condition. Then we find an available reduce action for application A_2, i.e. $\phi_{21}(1)$ is reduced from two to one, is effective for increasing the utility. After the corresponding reduce action is performed, the new allocation and placement strategy is

$$\phi_2 = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Then we go back to the migration process again and try to find a valid migration which increases the utility. By computing, we find no migration can be performed to increase its utility for application A_1 and A_2 while a valid migration, that ϕ_{31} is migrated from machine m_5 to m_2, can improve the performance for application A_3. After performing the migration, the new allocation strategy of A_3 is

$$\phi_3 = \begin{pmatrix} 1 & 2 & 1 & 1 & 1 \\ 2 & 1 & 2 & 2 & 1 \end{pmatrix}$$

Thus we repeat the overall algorithm until the global strategy has no change in the last cycle. The ultimate allocation and placement strategy of A_1,A_2 and A_3 is respectively

$$\phi_1 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

$$\phi_2 = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$\phi_3 = \begin{pmatrix} 1 & 2 & 1 & 1 & 1 \\ 2 & 1 & 2 & 2 & 1 \end{pmatrix}$$

In a word, each application maximizes its utility by our overall algorithm. The ultimate global strategy reaches a Nash equilibrium state, in which no application can change its strategy unilaterally to increase its utility. Although the Nash equilibrium can not ensure the global optimization, the result obtained by our algorithm is an effective and fair allocation strategy for each application.

9. CONCLUSIONS

In this paper, we address the dynamic resource allocation problem of multi-tiers web application in cloud. Considering the inherent complex interference between clients sharing the physical resource, we propose a comprehensive game-theoretic model, which captures the performance behavior of multi-tiers application and maps the client-client interaction to a dynamic pricing mechanism. The concept of Nash equilibrium is used to find an fair resource allocation.Firstly, a hill climb search algorithm is proposed to obtain the initial optimal allocation of IRAP. Then, based on the initial optimal allocation, we design an evolutionary algorithm to get the optimal placement of IVMPP. Finally, according to the solution of IRAP and IVMPP, we design a game theoretic allocation mechanism to allocate resources fairly for the original problem MRAPP. Moreover, we also prove that the ultimate result of our algorithm is actually a *Nash Allocation*.

Although we have taken into account several properties of a real cloud computing environment, we also assume that all the physical resources are not shut down which will violate the benefit of cloud providers in reality. Therefore, accounting for the conflict of benefit between the client and provider is a major part of our future work.

10. ACKNOWLEDGMENTS

This work is partially supported by the National Natural Foundation of China under Grant No. 61073028, 61021062; the National Basic Research Program of China (973) under Grant No. 2009CB320705; the Key Technology Research and Development Program of Jiangsu under Grant No. BE2010179; Jiangsu Natural Science Foundation under Grant No. BK2011510.

11. REFERENCES

[1] Amazon web services. `http://aws.amazon.com`.

[2] Google app engine. `http://code.google.com/appengine`.

[3] Windows azure. `http://www.microsoft.com/windowsazure`.

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[5] A. Azeez. Autoscaling web services on amazon ec2. Technical report, Department of Computer Science & Engineering University of Moratuwa, Sri Lanka.

[6] M. Bennani and D. Menasce. Resource allocation for autonomic data centers using analytic performance models. In *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, pages 229 –240, june 2005.

[7] A. Chandra, W. Gong, and P. Shenoy. Dynamic resource allocation for shared data centers using online measurements. In *Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '03, pages 300–301, New York, NY, USA, 2003. ACM.

[8] I. Cunha, J. Almeida, V. Almeida, and M. Santos. Self-adaptive capacity management for multi-tier virtualized environments. In *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*, pages 129 –138, 21 2007-yearly 25 2007.

[9] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou. The complexity of computing a nash equilibrium. *Commun. ACM*, 52(2):89–97, Feb. 2009.

[10] R. P. Doyle, J. S. Chase, O. M. Asad, W. Jin, and A. M. Vahdat. Model-based resource provisioning in a web service utility. In *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, pages 5–5, Berkeley, CA, USA, 2003. USENIX Association.

[11] M. W. D. C. P. Greg. Franks, Peter. Maly and A. Hubbard. Layered queueing network solver and simulator user manual. Technical report, Department of Systems and Computer Engineering in Carleton University, Canada, Dec 2005.

[12] A. B. J. Londono and S.-H. Teng. Collocation games and their application to distributed resource management. In *HotCloud*, 2009.

[13] G. Jung, K. Joshi, M. Hiltunen, R. Schlichting, and C. Pu. Generating adaptation policies for multi-tier applications in consolidated server environments. In *Autonomic Computing, 2008. ICAC '08. International Conference on*, pages 23 –32, june 2008.

[14] K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B. A. Huberman. Tycoon: An implementation of a distributed, market-based resource allocation system. *Multiagent Grid Syst.*, 1(3):169–182, Aug. 2005.

[15] G. Owen. *Game Theory*. Academic Press, 3 edition, 1995.

[16] T. Sandholm, K. Lai, J. Ortiz, and J. Odeberg. Market-based resource allocation using price prediction in a high performance computing grid for scientific applications. In *High Performance Distributed Computing, 2006 15th IEEE International Symposium on*, pages 132 –143, 0-0 2006.

[17] Y. Seung, T. Lam, L. Li, and T. Woo. Cloudflex: Seamless scaling of enterprise applications into the cloud. In *INFOCOM, 2011 Proceedings IEEE*, pages 211 –215, april 2011.

[18] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh. A cost-aware elasticity provisioning system for the cloud. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 559 –570, june 2011.

[19] B. Urgaonkar and A. Chandra. Dynamic provisioning

of multi-tier internet applications. In *Proceedings of the Second International Conference on Automatic Computing*, ICAC '05, pages 217–228, Washington, DC, USA, 2005. IEEE Computer Society.

[20] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi. An analytical model for multi-tier internet services and its applications. In *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '05, pages 291–302, New York, NY, USA, 2005. ACM.

[21] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood. Agile dynamic provisioning of multi-tier internet applications. *ACM Trans. Auton. Adapt. Syst.*, 3(1):1:1–1:39, Mar. 2008.

[22] D. Villela, P. Pradhan, and D. Rubenstein. Provisioning servers in the application tier for e-commerce systems. *ACM Trans. Internet Technol.*, 7(1), Feb. 2007.

[23] I. G. M. C. Virajith Jalaparti, Giang Nguyen. Cloud resource allocation games. Technical report, University of Illinois, Urbana-Champaign.

[24] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das. Utility functions in autonomic systems. In *ICAC*, pages 70–77, 2004.

[25] G. Wei, A. V. Vasilakos, Y. Zheng, and N. Xiong. A game-theoretic method of fair resource allocation for cloud computing services. *J. Supercomput.*, 54(2):252–269, Nov. 2010.