An Evolutionary Game Theoretic Approach to Adaptive and Stable Application Deployment in Clouds

Chonho Lee
University of Massachusetts, Boston
Boston, MA 02125, USA
chonho@cs.umb.edu

Junichi Suzuki
University of Massachusetts, Boston
Boston, MA 02125, USA
jxs@cs.umb.edu

Athanasios Vasilakos
University of Western Macedonia
GR 50100, Greece
vasilako@ath.forthnet.gr

Yuji Yamano
OGIS International, Inc.
San Mateo, CA 94404, USA
yyamano@ogis-international.com

Katsuya Oba
OGIS International, Inc.
San Mateo, CA 94404, USA
oba@ogis-international.com

ABSTRACT

This paper studies an evolutionary game theoretic mechanism for adaptive and stable application deployment in cloud computing environments. The proposed mechanism, called Nuage, allows applications to adapt their locations and resource allocation to the environmental conditions in a cloud (e.g., workload and resource availability) with respect to given performance objectives such as response time. Moreover, Nuage theoretically guarantees that every application performs an evolutionarily stable deployment strategy, which is an equilibrium solution under given environmental conditions. Simulation results verify this theoretical analysis; applications seek equilibria to perform adaptive and evolutionarily stable deployment strategies.

Categories and Subject Descriptors

C.2.4 [**Computer Systems Organization**]: Computer-Communication Networks—*Distributed Systems*; I.2 [**Computing Methodologies**]: Artificial Intelligence

Keywords

Cloud Computing, Adaptive and Stable Application Deployment, Evolutionary Game Theory

1. INTRODUCTION

One of key features in cloud computing (e.g., Infrastructure-as-a-Service and Platform-as-a-Service) environments is elastic scaling of their applications [1, 2, 3]. In order to provide this feature, they are required to perform dynamic application deployment for adjusting the locations and resource allocation of cloud applications [4, 5, 6, 7]. For example, they allocate different amounts of computing and networking resources (e.g., CPU time, memory/disk space and bandwidth) to each application according to the application's workload (i.e., the number of incoming messages). This allows applications to operate by balancing different performance objectives such as response time and resource consumption. (Resource consumption implies operational costs due to the pay-per-use models used in clouds.) Moreover, cloud computing environments relocate an application from one host to another and colocate multiple applications on the same host according to the resource availability on hosts. This allows applications to efficiently utilize resources and avoid the risk of host crashes due to resource scarcity.

This paper investigates two important properties in application deployment in clouds:

- *Adaptability*: allows applications to adapt their locations and resource allocation to workload and resource availability under given performance objectives.
- *Stability*: allows applications to seek stable adaptation decisions by minimizing oscillations (or non-deterministic inconsistencies) in decision making.

Nuage is an evolutionary game theoretic mechanism for adaptive and stable application deployment in clouds. This paper describes its design and evaluates its adaptability and stability. In Nuage, each application contains a set (or a population) of multiple players that represent different deployment strategies. Randomly-paired players repeatedly play games. Each game distinguishes a winning and a losing player with respect to performance objectives. The winner replicates itself and increases its share in the population. The loser is eliminated from the population. Through multiple games performed repeatedly in the population, the population state (or strategy distribution) changes. Through theoretical analysis, Nuage guarantees that the population state converges to an equilibrium where the population contains a dominant strategy. Nuage performs the dominant deployment strategy as the most rational strategy against given workload and resource availability.

Nuage theoretically proves that the population state is evolutionarily stable when it is on an equilibrium. An evolutionarily stable state is the state that, regardless of the initial population state, the population state always converges to. (A dominant strategy in the evolutionarily stable population state is called evolutionarily stable strategy.) Thanks to this property, Nuage guarantees that every application deterministically performs evolutionarily stable deployment strategy. Simulation results verify this theoretical analysis; applications seek equilibria to perform evolutionarily stable deployment strategies and adapt their locations and resource allocations to given workload and resource availability.

2. PROBLEM STATEMENT

This paper considers an application deployment problem where M hosts are available to operate N applications. Each application is designed and deployed as a set of three server software, following the three-tier application architecture [8]. Using a hypervisor such as Xen [9], each server is deployed on a virtual machine (VM) atop a host.

The goal of this problem is to find evolutionarily stable deployment strategies that deploy N applications (i.e., $N \times 3$ servers) on M hosts so that the applications adapt their locations and resource allocation to given workload and resource availability with respect to performance objectives. The placement of and the resource allocation to each application are conducted on a per-server (or per-VM) basis. This paper considers CPU time share (in percentage) as a resource assigned to each server (i.e., VM).

2.1 Application Architecture

Each application consists of the following three servers:

- Web server: accepts HTTP messages from application users, validates data in the messages and provides Web-based user interface for users.
- Application server: performs functional application logic and processes data transmitted from users.
- Database server: takes care of data access and storage.

Each message is sequentially processed from a Web server to a database server through an application server. A reply message is generated by the database server and forwarded in the reverse order toward a user. This paper assumes that different applications utilize different sets of servers. (Servers are not shared by different applications.) Users send different types of messages to different applications

A host can operate multiple VMs, each runs a server. Collocated VMs share resources available on their local host.

2.2 Performance Objectives

This paper considers the following performance objectives for each application to adapt its location and resource allocation. All objectives are to be minimized.

- *Response time*: The response time of an application for its users. It is estimated based on an M/M/1 queuing model [10], in which message arrivals follow a Poisson process and a server's service time is exponentially distributed to process incoming messages.
- *Resource consumption*: The total CPU time share (in percentage) assigned to three virtual machines in an application.
- *Distance*: The average distance between VMs in an application. It is computed as the hop count between hosts running the VMs.
- *Load balance*: The variance of workload (the number of incoming messages) among hosts running three VMs in an application.

The response time of an application (the i-th application) is estimated as follows.

$$R_i = T_i^{(s)} + T_i^{(w)} + T_i^{(d)} \tag{1}$$

$T_i^{(s)}$ denotes the time for the i-th application to process an incoming message from a user. It is computed as follows.

$$T_i^{(s)} = \sum_{v=1}^{3} T_{i,v}^{(s)} \tag{2}$$

$T_{i,v}^{(s)}$ denotes the service time of the v-th server in the i-th application. It indicates how long it takes for the server to process a message.

$T_i^{(w)}$ denotes the total waiting time for a message to be processed by three servers in the i-th application. It is computed as follows.

$$T_i^{(w)} = \frac{1}{\lambda_i} \sum_{v=1}^{3} \frac{\rho_{i,v} * \rho_{i,v}}{1 - \rho_{i,v}} \tag{3}$$

λ_i denotes the i-th application's message arrival rate (i.e., the number of messages the i-th application receives during the unit time). $\rho_{i,v}$ denotes the utilization of the v-th server in the i-th application. It is computed as follows.

$$\rho_{i,v} = \frac{\lambda_{i,v}}{C_{i,v}/T_{i,v}^{(s)}} \tag{4}$$

Note that $\lambda_i = \frac{\sum_{v=1}^{3} \lambda_{i,v}}{3}$. (Currently, $\lambda_i = \lambda_{i,1} = \lambda_{i,2} = \lambda_{i,3}$) $C_{i,v}$ denotes the CPU time share allocated to the v-th server in the i-th application.

$T_i^{(d)}$ is the total communication delay to transmit a message among servers in the i-th application. It is obtained with the size of a message and network bandwidth.

3. BACKGROUND: EVOLUTIONARY GAME THEORY

Game theory studies strategic selection of behaviors in interactions among rational players. In a game, given a set of strategies, each player strives to find a strategy that optimizes its own payoff depending on the others' strategy choices. Game theory seeks such strategies for all players as a solution, called *Nash equilibrium* (NE), where no players can gain extra payoff by unilaterally changing his strategy.

Evolutionary game theory (EGT) is an application of game theory to biological contexts to analyze population dynamics and stability in biological systems. In EGT, games are played repeatedly by players randomly drawn from the population [11, 12]. In general, EGT considers two major evolutionary mechanisms: *mutation*, which injects varieties on genes, and *selection*, which favors some varieties over others based on their fitness to the environment. Mutation is considered in the notion of evolutionarily stable strategies (ESS), which is a refinement of NE. Selection is considered in the replicator dynamics (RD) model.

3.1 Evolutionarily Stable Strategies

ESS is a key concept in EGT. A population following such a strategy is invincible. Specifically, suppose that the initial population is programmed to play a certain pure or mixed strategy x (the incumbent strategy). Then, let a small population share of players $\epsilon \in (0, 1)$ play a different pure or mixed strategy y (the mutant strategy). Hence, if a player is drawn to play the game, the probabilities that its opponent plays the incumbent strategy x and the mutant strategy y are $1 - \epsilon$ and ϵ, respectively. The player's payoff of such a game is the same as that of a game where the player plays the mixed strategy $w = \epsilon y + (1 - \epsilon)x$. The payoffs of players with strategies x and y given that the opponent adopts strategy w are denoted by $U(x, w)$ and $U(y, w)$, respectively.

Definition 1. *A strategy x is called evolutionarily stable if, for every strategy $y \neq x$, a certain $\bar{\epsilon} \in (0, 1)$ exists, such that the inequality*

$$U(x, \epsilon y + (1 - \epsilon)x) > U(y, \epsilon y + (1 - \epsilon)x) \quad (5)$$

holds for all $\epsilon \in (0, \bar{\epsilon})$.

In the special case where the payoff function is linear, $U(x, w)$ and $U(y, w)$ can be written as the expected payoffs for players with strategies x and y, and Equation (5) yields

$$(1 - \epsilon)U(x, x) + \epsilon U(x, y) > (1 - \epsilon)U(y, x) + \epsilon U(y, y) \quad (6)$$

If ϵ is close to zero, Equation (6) yields either

$$\begin{gathered} U(x, x) > U(y, x), \quad \text{or} \\ U(x, x) = U(y, x) \ \text{ and } \ U(x, y) > U(y, y) \end{gathered} \quad (7)$$

Hence, it becomes obvious that an ESS must be a NE; otherwise, Equation (7) do not hold.

3.2 Replicator Dynamics

The replicator dynamics, first proposed by Taylor [13], specifies how population shares associated with different pure strategies evolve over time. In replicator dynamics players are programmed to play only pure strategies. To define the replicator dynamics, consider a large but finite population of players programmed to play pure strategy $k \in K$, where K is the set of strategies. At any instant t, let $\lambda_k(t) \geq 0$ be the number of players programmed to play pure strategy k. The total population of players is given by $\lambda(t) = \sum_{k \in K} \lambda_k(t)$. Let $x_k(t) = \lambda_k(t)/\lambda(t)$ be the fraction of players using pure strategy k at time t. The associated population state is defined by the vector $\mathbf{x}(t) = [x_1(t), \cdots, x_k(t), \cdots, x_K(t)]$. Then, the expected payoff of using pure strategy k given that the population is in state $\mathbf{x}$ is $U(k, \mathbf{x})$ and the *population average payoff*, that is the payoff of a player drawn randomly from the population, is $U(\mathbf{x}, \mathbf{x}) = \sum_{k=1}^{K} x_k \cdot U(k, \mathbf{x})$. Suppose that payoffs are proportional to the reproduction rate of each player and, furthermore, that a strategy profile is inherited. This leads to the following dynamics for the population shares x_k

$$\dot{x}_k = x_k \cdot [U(k, \mathbf{x}) - U(\mathbf{x}, \mathbf{x})] \quad (8)$$

where x_k is the time derivative of x_k. The equation states that populations with better (worse) strategies than average grow (shrink). However, there are cases when even *a strictly dominated strategy* may gain more than average. Hence, it is not a priori clear whether if such strategies get wiped out in the replicator dynamics. The following theorem answers this question [11]:

Theorem 1. *If a pure strategy k is strictly dominated then $\xi_k(t, x^0)_{t \to \infty} \to 0$, where $\xi_k(t, x^0)$is the population at time t and x^0 is the initial state.*

On the other hand, it should be noted that the ratio x_k/x_ℓ of two population shares $x_k > 0$ and $x_\ell > 0$ increases with time if the strictly dominated strategy k gains a higher payoff than the strictly dominated strategy ℓ. This is a direct result of Equation (8) and may be expressed analytically via

$$\frac{d}{dt}\left[\frac{x_k}{x_\ell}\right] = [U(k, \mathbf{x}) - U(\ell, \mathbf{x})]\frac{x_k}{x_\ell} \quad (9)$$

From Equation (9), it is evident that even suboptimal strategies could temporarily increase their share before being wiped out in the long run. However, there is a close connection between NE and the steady states of the replicator dynamics, which is states where the population shares do not change their strategies over time. Thus, since in NE all strategies have the same average payoff, every NE is a steady state. The reverse is not always true: Steady states are not necessarily NE, e.g., any state where all players use the same pure strategy is a steady state, but, it is not stable [11].

In this paper, a single fixed-sized population model is used; also, discrete time (i.e., generational) model is assumed.

4. NUAGE

4.1 A Design of Evolutionary Game in Nuage

Nuage executes an independent evolutionary game in each of N different applications' populations, $\{AP_1, AP_2, ..., AP_N\}$. An application population AP_i contains M players $\{p_{i,1}, p_{i,2}, ..., p_{i,M}\}$. A player has its own strategy $S(p_{i,j})$ that represents a deployment of three virtual servers v_i^t corresponding to an application i, where $t \in \{1, 2, 3\}$ (e.g., 1 for Web, 2 for App, and 3 for DB servers). Each virtual server is deployed on a host $h \in H$. The strategy specifies *placements* and *resource allocations* of the virtual servers. A placement indicates which host a virtual server is deployed on. An allocation indicates CPU time share assigned to the virtual server. Thus, a strategy is described as a set of pairs of the placement and allocation for three virtual servers v_i^t where $t \in \{1, 2, 3\}$ and implemented as

$$S(p_{i,j}) = \{(h_i^1, \ell_i^1), (h_i^2, \ell_i^2), (h_i^3, \ell_i^3)\} \quad (10)$$

where h_i^t denotes a host ID running a virtual server v_i^t for an application i, and ℓ_i^t denotes CPU time share (%) assigned to the virtual server v_i^t for an application i.

A game is repeatedly performed between randomly paired players in an application population. According to their fitness, a winning/losing player increases/decreases its subpopulation in an application population. A strategy of a player whose subpopulation is the largest is used as the current deployment of an application. Figure 1 shows example strategies for two different applications (a_1 and a_2).

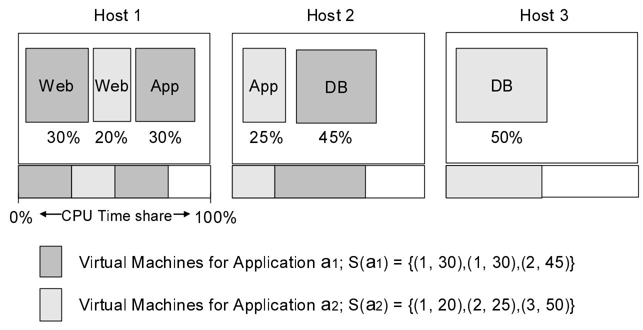

Host 1
Host 2
Host 3
Web
Web
App
App
DB
DB
30%
20%
30%
25%
45%
50%
0% ←CPU Time share→ 100%
Virtual Machines for Application a1; S(a1) = {(1, 30),(1, 30),(2, 45)}
Virtual Machines for Application a2; S(a2) = {(1, 20),(2, 25),(3, 50)}

Figure 1: An example allocation of two applications

A Nuage state can be described as a *deployment state* that indicates an placements and allocations for all applications (i.e., $3N$ virtual servers). It is denoted as a ($3N$ x $|H|$)-matrix $X = [x_{rc}]$ where x_{rc} is an allocation of a virtual server v_r at a host c. The virtual server ID, r, is given by $r = 3 * i + t$ where i is an application ID, and t is an index of virtual servers $\in \{1, 2, 3\}$.

$$X = \begin{bmatrix} 30 & 0 & 0 \\ 20 & 0 & 0 \\ 0 & 45 & 0 \\ 30 & 0 & 0 \\ 0 & 25 & 0 \\ 0 & 0 & 50 \end{bmatrix} \; where \; \sum_{r=1}^{3N} x_{rc} \leq 100 \tag{11}$$

4.2 A Procedure of Nuage

Applications evolve through generations by changing their strategies (i.e., placements and resource allocations) and improve their objective values (e.g., response time for users). At each generation, each application repeatedly performs games between randomly-paired players in the application population. A winning player will make its copy, and a losing player will be removed from the population. Then, a mutation operation is performed on each copied player at a certain probability to change its strategy. The mutation occurs at one of virtual servers with (h_i^t, ℓ_i^t). One of available hosts is randomly assigned to h_i^t, and the value of ℓ_i^t is assigned based on a normal distribution $G(\mu, \sigma) = G(\ell_i^t, 1)$.

A player wins/loses against the opponent according to their *fitness*. In this paper, the fitness is given by a *domination relationship*. The domination relationship is determined based on *objectives* described in Section 2.2 and their *priority*. By definition, it is said that Player A *dominates* Player B if all objective values of A is smaller than that of B. It is said that Player A is *superior* to Player B if the number of dominating objectives of A is larger than that of B. If those numbers for A and B are the same, then they consider the pre-defined priority of objectives.

Figure 2 shows a pseudocode of the mechanism to explain how Nuage works. InitializePopulation(AP) initializes all application populations by assigning randomly chosen strategies to players; Randomize(O) permutates the elements of a set O, which is a set of indexes of applications; Select(AP_i) retrieves two players randomly from an application population AP_i; and PerformGame(p_x, p_y) determines winning/losing players by evaluating their domination relationship.

```
Nuage()
 // AP: a set of application populations
 // O: a permutation ordering of applications' indexes
 // W: a set of winning players, R: a set of the mutated
 // $p_x$: A player with a strategy $x$
main
 InitializePopulation($AP$)
 $O \leftarrow (1, 2, ..., N)$
 while (the termination condition is not satisfied)
   do
     $O \leftarrow$ Randomize($O$)
     for $r \leftarrow 0$ to $N$
       do
         $i \leftarrow O(r)$
         $W, R \leftarrow \phi,\ M \leftarrow |AP_i|/2$
         for $j \leftarrow 0$ to $M$
           do
             $\{p_1, p_2\} \leftarrow$ Select($AP_i$)
             $AP_i \leftarrow AP_i - \{p_1, p_2\}$
             $winner \leftarrow$ PerformGame($p_1, p_2$)
             $W \leftarrow W \cup winner$
             $R \leftarrow R \cup$ Mutate($winner$)
         $AP_i \leftarrow W \cup R$
```

Figure 2: Evolutionary Games in Nuage

5. STABILITY ANALYSIS

This section analyzes the stability of Nuage by showing that an application population state converges to an evolutionarily stable state (or an asymptotically stable state) in three steps: (1) The dynamics of the population state change over time is formalized as a set of differential equations, (2) The proposed evolutionary game has equilibrium points, (3) The equilibrium points are asymptotically stable. First, in order to construct the differential equations, following terminologies and variables are defined.

- S denotes a set of strategies. A strategy, $s \in S$, consists of pairs of a placement and allocation for three virtual servers as described in Section 4.1. S^* denotes a set of strategies that appear in an application population.
- M denotes a population size. $M = \sum_{s \in S^*} n_s$ where n_s is the number of players with a strategy s.
- $X(t)$ denotes a population state at time t. $X(t) = \{x_1(t), x_2(t), \cdots, x_{|s^*|}(t)\}$ where x_s is the population share of players with a strategy s ($x_s = \frac{n_s}{|S^*|}$; $\sum_{s \in S^*} x_s = 1$).
- F_s is the fitness of a player with a strategy b.
- p_k^s denotes the probability that a player with a strategy s is replicated by winning a game against the player with a strategy k. It is computed by $p_k^s = x_s \cdot \phi(F_s - F_k)$ where $\phi(F_b - F_k)$ is the conditional probability that the fitness of a player with a strategy s is larger than that of a strategy k.

How players with a strategy s change their population share is considered as the sum of difference between the number of players which are replicated (win) and eliminated (lose) at a time; then it is formalized as follows (using a brevity $c_{sk} = \phi(F_s - F_k) - \phi(F_k - F_s)$).

$$\begin{aligned} \dot{x}_s &= \sum_{k \in S^*, k \neq s} \{x_k p_k^s - x_s p_s^k\} \\ &= x_s \sum_{k \in S^*, k \neq s} x_k \{\phi(F_s - F_k) - \phi(F_k - F_s)\} \\ &= x_s \sum_{k \in B, k \neq s} x_k \cdot c_{sk} \end{aligned} \quad (12)$$

Theorem 2. *If a player with a strategy s is strictly dominated, then $x_s(t) \to 0$ as $t \to \infty$.*

In game theory, it is said that a strategy is strictly dominant if, regardless of what any other players select, a player with the strategy gains a strictly higher fitness than any others. If a player has a strictly dominant strategy, than it is always better than any others in terms of fitness (i.e., a domination relationship). It increases its population share and occupies a population over time. So, if a player is strictly dominated, then the player disappears in a population over time.

Theorem 3. *The population state of an application population converges to an equilibrium.*

Proof. It is true that, players with different strategies have different domination factors under the same network conditions. In other words, under the particular network conditions, only one player has the highest domination rank among the others. Assume that $F_1 > F_2 > \cdots > F_{|s^*|}$, and by Theorem 1, a population state eventually converges to $X(t) = \{x_1(t), x_2(t), \cdots, x_{|s^*|}(t)\} = \{1, 0, \cdots, 0\}$ as an equilibrium. Differential equations should satisfy the constraint $\sum_{s \in s^*} x_s = 1$. $\square$

Theorem 4. *The equilibrium of Nuage is evolutionarily stable (i.e., asymptotically stable).*

Proof. At the equilibrium where $X = \{1, 0, \cdots, 0\}$, a set of differential equations can be rewritten in the downsized by substituting $x_1 = 1 - x_2 - \cdots - x_{|s^*|}$

$$\dot{z}_s = z_s[c_{s1}(1 - z_s) + \sum_{i=2, i \neq s}^{|s^*|} z_i \cdot c_{si}] \quad (13)$$
$$where\ s, k = 2, ..., |s^*|$$

where $Z(t) = \{z_2(t), z_3(t), \cdots, z_{|s^*|}(t)\}$ denotes the corresponding downsized population state, which is an equilibrium $Z_{eq} = \{0, 0, \cdots, 0\}$ of $(|s^*| - 1)$-dimension based on Theorem 1.

To verify that a state at the equilibrium is an asymptotically stable state, show that all the Eigenvalues of Jaccobian matrix of the downsized population state has negative Real parts. The elements of Jaccobian matrix J are

$$\begin{aligned} J_{bk} &= \left[\frac{\partial \dot{z}_b}{\partial z_k}\right]_{|Z = Z_{eq}} \\ &= \left[\frac{\partial z_s[c_{s1}(1 - z_s) + \sum_{i=2, i \neq s}^{|s^*|} z_i \cdot c_{si}]}{\partial z_k}\right]_{|Z = Z_{eq}} \end{aligned} \quad (14)$$
$$where\ s, k = 2, ..., |s^*|$$

Therefore, Jaccobian matrix J is given by

$$J = \begin{bmatrix} c_{21} & 0 & \cdots & 0 \\ 0 & c_{31} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & c_{M1} \end{bmatrix} \quad (15)$$

where $c_{21}, c_{31}, \cdots, c_{|s^*|1}$ are the Eigenvalues of J. According to Theorem 2, $c_{s1} = -\phi(F_1 - F_s) < 0$ for every s; therefore, $Z_{eq} = \{0, 0, \cdots, 0\}$ is asymptotically stable. $\square$

6. EVALUATION

This section evaluates the proposed Nuage in simulation studies and shows how applications change their objective values over time by changing their strategies (i.e., application deployments).

6.1 Simulation Configurations

This simulation considers 5 types of applications on 5 hosts (or machines) which are fully-connected or linearly-connected. When a distance objective is considered, the linearly-connected hosts are used. Table 1 shows message arrival rate (# of requests/sec) and service time (sec) for the applications, which are pre-defined based on Zipf's law [14, 15]. The service time for applications are defined as 0.1, 0.15, 0.2, 0.25, and 0.3 sec respectively. An entry for the service time indicates the expected processing time for a single message on a virtual server.

Table 1: Message Arrival Rate (# of messages/sec) and Service Time (sec)

Application type	1	2	3	4	5
Arrival rate	110	70	40	20	10
Web server	0.02	0.02	0.04	0.04	0.08
App server	0.03	0.08	0.04	0.13	0.11
DB server	0.05	0.05	0.12	0.08	0.11

Application 1 is assumed to be an application whose computational operations and data management operations are light. Application 2 is assumed to be an application whose computational operations are heavier than Application 1. Application 3 is assumed to be an application whose data management operations are heavier than Application 1. Application 4 is assumed to be an application whose operations is computationally more intensive than Application 2. Application 5 is assumed to be an application whose all operations are heavy.

For the proposed evolutionary game described in Section 4.2, a population size for each application is set to 50, a mutation probability is set to 0.02, and the communication delay $T^{(d)}$ is set to 0.05 sec. All simulation results are average results of 100 independent runs.

6.2 Simulation Results

Simulation results shows how applications' strategies (i.e., deployments) impact to objective values. Traces of the objective values are shown in Figure 4 - 7. Each subfigure shows traces of objective values when applications consider

Table 2: Objective Combinations

Combination	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
Response time	✓				✓	✓	✓	✓
Resource consumption		✓			✓	✓	✓	✓
Distance			✓			✓		✓
Load balancing				✓			✓	✓

a particular combination of those objectives. The evaluated objective combinations are described in Table 2.

First, in order to investigate how applications change their strategies (i.e., deployments) during their evolutionary games, Figure 3 shows population states of applications over generations in a case C_8. Different lines represent the normalized number of agents with different strategies. Each application selects a strategy of the agent whose population share is the largest in the application population (Labeled numbers in the figure represent strategy IDs). The right bottom subfigure in Figure 3 shows a deployment state at generation 100 in a case C_8. For example, App1 selects a strategy $S_{62} = \{(1,6)(1,7)(1,10)\}$ at generation 100.

Figure 4 shows the average response time for applications. The response time successfully decreases over generations for applications by changing their deployments. Especially, cases C_1 and C_3 result in shorter response time than the others. Placing virtual servers at the same host contributes to reduce response time. In cases C_5, C_7, C_8, the response time does not decrease as C_1 and C_3 do due to multiple objectives. In a case C_6, the response time is relatively shorter than C_5, C_7, C_8 because it considers a distance objective.

Figure 5 shows the resource consumption (i.e., assigned CPU time share in %) for applications. Applications try to reduce resource consumption as much as they can for the efficient use of resources. However, in cases C_1, C_3, C_4, applications does not care for minimizing resource consumption and they require more resource to process their messages.

Figure 6 shows the average distance (hop counts) among hosts running virtual servers for applications. These results are evaluated in linearly-connected hosts. Cases C_1 and C_3 result in shorter distance than the others similar to the response time results. Placing virtual servers at the same hosts or closer to each other contributes to reduce response time. A case C_6 shows a relatively shorter distance compared to C_5, C_7, C_8 because it considers a distance objective.

Figure 7 evaluates load balancing among hosts running virtual servers for applications. Load balancing index (LBI) is computed as the variance of workload (the number of user messages) among hosts running virtual servers. The smaller is the better. A case C_4 successfully reduces LBI to around 20, which is smaller than the other cases (around 24-28). All the other cases could not reduce LBI as C_4 although C_8 shows a relatively smaller LBI.

7. RELATED WORK

Several game theoretic approaches have been proposed for adaptive application placement in clouds [7, 16, 17]. They formulate equilibria in application placement and use greedy algorithms to seek equilibrium solutions under multiple performance objectives. This means they do not focus on stability in application placement. In contrast, Nuage employs an evolutionary game theoretic approach and theoretically guarantees its stability in finding adaptive strategies for resource allocation as well as application placement under multiple performance objectives. Nuage also considers multi-tier application architecture, while [7, 16, 17] do not .

[18, 19] and [20] seek the optimal placement of a single application and multi-server applications, respectively, under a single performance objective. It is out of their scope to seek equilibrium solutions and attain stability in finding solutions. In contrast, Nuage considers the placement of and the resource allocation to multiple multi-server applications. It is designed to find stable equilibrium solutions.

[21, 22, 23] propose task/job scheduling mechanisms that assign resources and orders to process tasks, and they assume that the tasks are independent each other. Those algorithms can still work with many loose coupling service-integrated applications. However, most cloud-based applications consist of multiple subtasks and require communications among tasks [16]. In this paper, Nuage considers communications among virtual servers where tasks (i.e., messages) are required to be sequentially processed on.

8. CONCLUSIONS

This paper describes Nuage, an evolutionary game theoretic mechanism for adaptive and stable application deployment in clouds. Nuage theoretically guarantees that every application performs an evolutionarily stable deployment strategy, which is an equilibrium solution under given workload and resource availability. Simulation results verify this theoretical analysis; applications seek equilibria to perform adaptive and evolutionarily stable deployment strategies.

As future work, the authors of the paper plan to carry out extended simulation studies that consider not only CPU time share but also memory space and network bandwidth as resources. It is also planned to compare Nuage with existing optimization algorithms in order to evaluate the optimality as well as stability in Nuage.

9. REFERENCES

[1] A. Weiss. Computing in the clouds. *ACM netWorker Magazine*, 11(4), 2007.

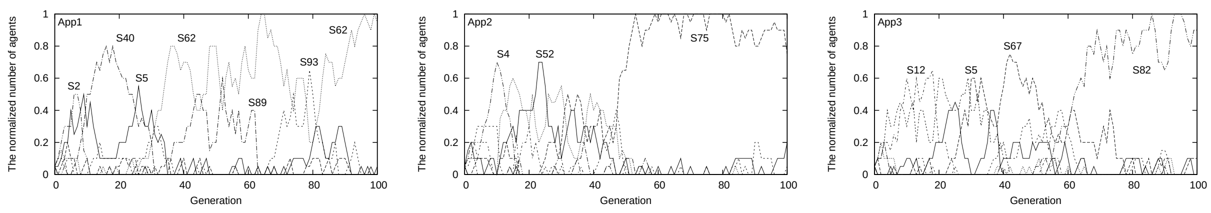
[2] J. Varia. Cloud architectures. Technical report, Amazon Web Services, 2007.

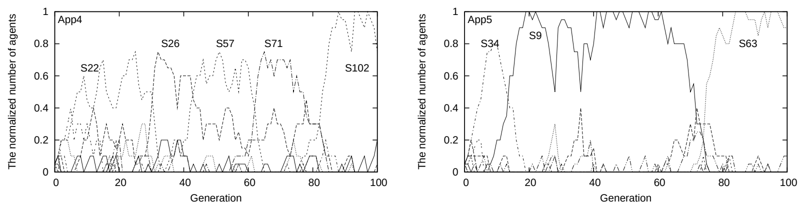
[3] G. Boss, P. Malladi, D. Quan, L. Legregni, and H. Hall. Cloud computing. Technical report, IBM High Performance On Demand Solutions, 2007.

[4] M.Caramia and S.Giordani. Resource allocation in grid computing: an economic model. *Transactions on Computer Research*, 3(1), 2008.

[5] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova. Resource allocation using virtual clusters. In *Proc. of IEEE/ACM Int'l Symposium on Cluster Computing and the Grid*, May 2009.

[6] C. A. Yfoulis and A. Gounaris. Honoring SLAs on cloud computing services: A control perspective. In





$$\begin{array}{ll|ccccc} & & \text{Host1} & \text{Host2} & \text{Host3} & \text{Host4} & \text{Host5} \\ \text{App1:} & \text{VM1} & 6 & . & . & . & . \\ & \text{VM2} & 7 & . & . & . & . \\ & \text{VM3} & 10 & . & . & . & . \\ \text{App2:} & \text{VM1} & . & . & . & 6 & . \\ & \text{VM2} & . & . & . & 12 & . \\ & \text{VM3} & . & . & . & 9 & . \\ \text{App3:} & \text{VM1} & . & . & 8 & . & . \\ & \text{VM2} & . & . & 9 & . & . \\ & \text{VM3} & 15 & . & . & . & . \\ \text{App4:} & \text{VM1} & . & 9 & . & . & . \\ & \text{VM2} & . & . & . & . & 16 \\ & \text{VM3} & . & . & . & . & 8 \\ \text{App5:} & \text{VM1} & . & 13 & . & . & . \\ & \text{VM2} & . & 15 & . & . & . \\ & \text{VM3} & . & 15 & . & . & . \end{array}$$

Figure 3: Population states of applications and the deployment state at generation 100 ('.' means 0)

Proc. of EUCA/IEEE European Control Conference, August 2009.

[7] S. U. Khan and C. Ardil. Energy efficient resource allocation in distributed computing systems. In *Proc. of WASET Int'l Conference on Distributed, High-Performance and Grid Computing*, August 2009.

[8] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi. An analytical model for multi-tier internet services and its applications. In *Proc. of ACM Int'l Conference on Measurement and Modeling of Computer Systems*, June 2005.

[9] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proc. of ACM symposium on operating systems principles*, October 2003.

[10] R. B. Cooper. *Introduction to Queueing Theory.* North-Holland Elsevier, 1981.

[11] J. W. Weibull. *Evolutionary Game Theory.* MIT Press, 1996.

[12] M. A. Nowak. *Evolutionary Dynamics: Exploring the Equations of Life.* Harvard University Press, 2006.

[13] P. Taylor and L. Jonker. Evolutionary stable strategies and game dynamics. *Elsevier Mathematical Biosciences*, 40(1), 1978.

[14] R. Perline. Zipf's law, the central limit theorem, and the random division of the unit interval. *Physical Review E*, 54(1), 1996.

[15] J. Tatemura, W-P. Hsiung, and W-S. Li. Acceleration of web service workflow execution through edge computing. In *Proc. of Int'l World Wide Web Conference*, May 2003.

[16] G. Wei, A. V. Vasilakos, Y. Zheng, and N. Xiong. A game-theoretic method of fair resource allocation for cloud computing services. *Journal of Supercomputing*, Accepted, 2009.

[17] N. Doulamis, A. Doulamis, A. Litke, A. Panagakis, T. Varvarigou, and E. Varvarigos. Adjusted fair scheduling and non-linear workload prediction for qos guarantees in grid computing. *Elsevier Computer Communications*, 30(3), 2007.

[18] C. S. Xiaoyun, X. Zhu, and H. Crowder. A mathematical optimization approach for resource allocation in large scale data centers. Technical report, HP Labs Palo Alto, 2002.

[19] A. Zhang, P. Santos, D. Beyer, and H k. Tang. Optimal server resource allocation using an open queuing network model of response time. Technical report, HP Labs Palo Alto, 2002.

[20] S. Borst, O. Boxma, J. F. Groote, and S. Mauw. Task allocation in a multi-server system. *Journal of Scheduling*, 6(5), 2003.

[21] R. P. Doyle. Model-based resource provisioning in a web service utility. In *Proc. of USENIX Symposium on Internet Technologies and Systems*, March 2003.

[22] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in grid computing. *Journal of Concurrency and Computation*, 14(13), 2002.

[23] A. Byde, A. Byde, M. Salle, M. Salle, and C. Bartolini. Market-based resource allocation for utility data centers. Technical report, HP Labs Bristol, 2003.

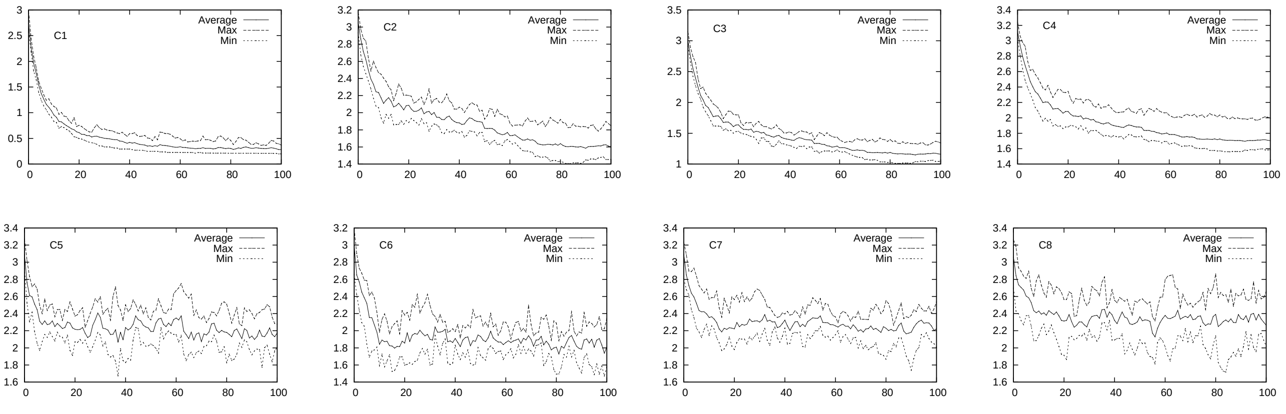


Figure 4: Average response time for users (sec) over 100 generations (x-axis)

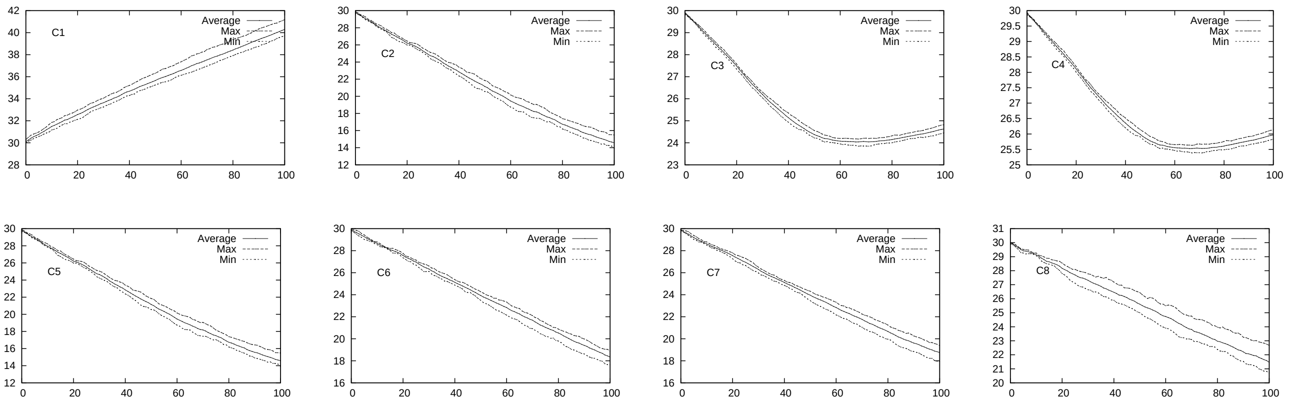


Figure 5: Resource consumption (%) over 100 generations (x-axis)

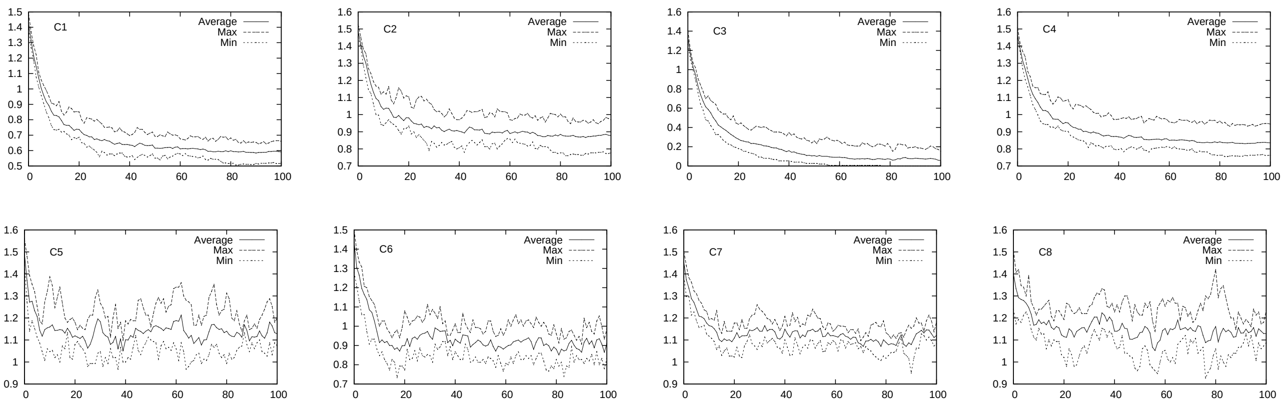


Figure 6: Average distance among hosts running virtual servers (hop counts) over 100 generations (x-axis)

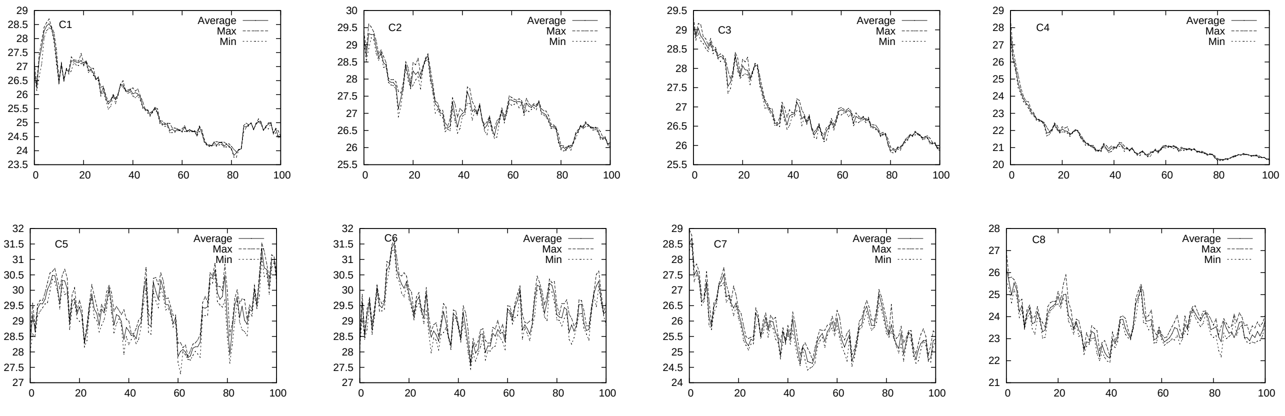


Figure 7: Load balancing index (LBI) over 100 generations (x-axis)