International Journal of Software Engineering and Knowledge Engineering

LEVERAGING ACTIVE-GUIDED EVOLUTIONARY GAMES FOR ADAPTIVE AND STABLE DEPLOYMENT OF DVFS-AWARE CLOUD APPLICATIONS

YI CHENG REN

Department of Computer Science
University of Massachusetts Boston
Boston, MA 02125, USA
yiren001@cs.umb.edu

JUNICHI SUZUKI

Department of Computer Science
University of Massachusetts Boston
Boston, MA 02125, USA
jxs@cs.umb.edu
http://www.cs.umb.edu/~jxs/

SHINGO OMURA

OGIS International, Inc.
San Mateo, CA 94402, USA
omura@ogis-international.com

RYUICHI HOSOYA

OGIS International, Inc.
San Mateo, CA 94402, USA
hosoya@ogis-international.com

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

This paper proposes and evaluates a multiobjective evolutionary game theoretic framework for adaptive and stable application deployment in clouds that support dynamic voltage and frequency scaling (DVFS) for CPUs. The proposed algorithm, called AGEGT, aids cloud operators to adapt the resource allocation to applications and their locations according to the operational conditions in a cloud (e.g., workload and resource availability) with respect to multiple conflicting objectives such as response time, resource utilization and power consumption. In AGEGT, evolutionary multiobjective games are performed on application deployment strategies (i.e., solution candidates) with an aid of guided local search. AGEGT theoretically guarantees that each application performs an evolutionarily stable deployment strategy, which is an equilibrium solution under given operational conditions. Simulation results verify this theoretical analysis; applications seek equilibria to perform adaptive and evolutionarily stable deployment strategies. AGEGT allows applications to successfully leverage DVFS to balance their response

time, resource utilization and power consumption. AGEGT gains performance improvement via guided local search and outperforms existing heuristics such as first-fit and best-fit algorithms (FFA and BFA) as well as NSGA-II.

Keywords: Cloud computing; evolutionary game theory; multiobjective optimization, guided local search

1. Introduction

It is a challenging issue for cloud operators to deploy applications so that the applications can keep expected levels of performance (e.g. response time) while maintaining their utilization of resources (e.g. CPUs and bandwidth) and power consumption. In order to ensure these requirements, they are required to dynamically (re-)deploy applications by adjusting their locations and resource allocation according to various operational conditions such as workload and resource availability. This paper investigates two important properties of application deployment in clouds:

- *Adaptability*: Adjusting the locations of and resource allocation for applications according to operational conditions (e.g., workload and resource availability) with respect to given objectives such as response time, resource utilization and power consumption.
- *Stability*: Minimizing oscillations (non-deterministic inconsistencies) in making adaptation decisions.

AGEGT is an evolutionary game theoretic framework for adaptive and stable application deployment in clouds that support dynamic voltage and frequency scaling (DVFS) for CPUs. This paper describes its design and evaluates its adaptability and stability. In AGEGT, each application maintains a set (or a population) of deployment strategies, each of which indicates the location of and resource allocation for that application. AGEGT repeatedly performs evolutionary multiobjective games on deployment strategies and evolves them over generations with respect to conflicting objectives. In each generation, AGEGT runs *active-guided mutation*, which alters deployment strategies based on the guided local search (GLS) algorithm [30]. It records inferior deployment strategies as *penalties* through generations and uses the penalties to help strategies escape from local optima and gain performance improvement.

AGEGT theoretically guarantees that, through a series of evolutionary games between deployment strategies, the population state (i.e., the distribution of strategies) converges to an evolutionarily stable equilibrium, which is always converged to regardless of the initial state. (A dominant strategy in the evolutionarily stable population state is called an *evolutionarily stable strategy*.) In this state, no other strategies except an evolutionarily stable strategy can dominate the population. Given this theoretical property, AGEGT aids each application to operate at equilibria by using an evolutionarily stable strategy for application deployment in a deterministic (i.e., stable) manner.

Simulation results verify this theoretical analysis; applications seek equilibria to perform evolutionarily stable deployment strategies and adapt their locations and resource allocations to given operational conditions. AGEGT allows applications to successfully leverage DVFS to balance their response time performance, resource utilization and power consumption. AGEGT's performance is evaluated in comparison to existing heuristic algorithms. Simulation results demonstrate that AGEGT outperforms a well-known multiobjective evolutionary optimization algorithm, NSGA-II [7] by 25% in deployment quality while maintaining 98% higher stability (lower oscillations) in performance across different simulation runs. Moreover, AGEGT outperforms two well-known heuristics, first-fit and best-fit algorithms (FFA and BFA), which have been widely used for adaptive cloud application deployment [2, 11, 18, 19].

2. Problem Statement

This section formulates an application deployment problem where M hosts are available in a cloud data center to operate N applications. Each application is designed with a set of server software, following a three-tier application architecture [23, 28] (Fig. 1). Using a certain hypervisor such as Xen [1], each server is assumed to run on a virtual machine (VM) atop a host. A host can operate multiple VMs. They share resources available on their local host. Each host is assumed to be equipped with a multi-core CPU.

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. (Fig. 1). This paper assumes that different applications utilize different sets of servers. (Servers are not shared by different applications.) And each host runs multi cores processor to allocate differents applications.

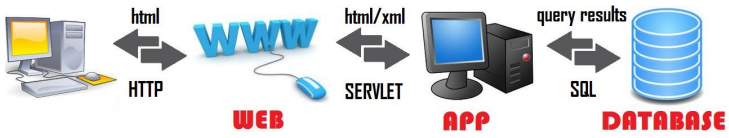


Fig. 1. Three Tiers of Web, Application and Database Servers

The goal of this problem is to find evolutionarily stable strategies that deploy N applications (i.e., $N \times 3$ VMs) on M hosts so that the applications adapt their locations and resource allocation to given workload and resource availability with respect to the four objectives described below. Every objective is computed on an application by application basis.

- *CPU allocation* (f_C): A certain CPU time share (in percentage) is allocated to each VM. (The CPU share of 100% means that a CPU core is fully allo-

cated to a VM.) It represents the upper limit for the VM's CPU utilization. This objective is computed as the sum of CPU shares allocated to three VMs of an application.

$$f_C = \sum_{t=1}^{3} c_t \tag{1}$$

c_t denotes the CPU time share allocated to the t-th tier server in an application.

- *Bandwidth allocation* (f_B): A certain amount of bandwidth (in bits/second) is allocated to each VM. It represents the upper limit for the VM's bandwidth consumption. This objective is computed as the sum of bandwidth allocated to three VMs of an application.

$$f_B = \sum_{t=1}^{3} b_t \tag{2}$$

b_t denotes the amount of bandwidth allocated to the t-th tier server in an application.

- *Response time* (f_{RT}): This objective indicates the time required for a message to travel from a web server to a database server. It is computed as follows.

$$f_{RT} = T^p + T^w + T^c \tag{3}$$

T^p denotes the total time for an application to process an incoming message from a user at three servers. T^w is the waiting time for a message to be processed at servers. T^c denotes the total communication delay to transmit a message between servers. T^p, T^w and T^c are estimated with the $M/M/M$ queuing model, in which message arrivals follow a Poisson process and a server's message processing time is exponentially distributed.

T^p is computed as follows where T_t^p denotes the time required for the t-th tier server to process a message.

$$T^p = \sum_{t=1}^{3} T_t^p \tag{4}$$

T^w is computed as follows.

$$T^w = \frac{1}{\lambda} \sum_{t=1}^{3} \rho_0 \frac{a_t^O}{O!} \frac{\rho_t}{(1-\rho_t)^2} \tag{5}$$

$$\text{where } a_t = \lambda_t \frac{T_t^p}{c_t \cdot q_t / q_{max}},\ \rho_t = \frac{a_t}{O},\ \rho_0 = \left(\sum_{n=0}^{O-1} \frac{\rho_t^n}{n!} + \frac{\rho_t^O}{O!} \frac{1}{1-\rho_t/O} \right)^{-1}$$

λ denotes the message arrival rate for an application (i.e., the number of messages the application receives from users in the unit time). Note that $\lambda = \frac{1}{3}\sum_{t=1}^{3}\lambda_t$ where λ_t is the message arrival rate for the t-th tier server in the application. Currently, $\lambda = \lambda_1 = \lambda_2 = \lambda_3$. ρ_t denotes the utilization of a CPU core that the t-th tier server resides on. q_{max} is is the maximum CPU frequency. q_t is the frequency of a CPU core that the t-th tier server resides on. O is the total number of cores that a CPU contains.

T^c is computed as follows.

$$T^c = \sum_{t=1}^{2} T^c_{t\rightarrow t+1} \approx \sum_{t'=2}^{3} \frac{B \cdot \lambda_{t+1}}{b_t} \tag{6}$$

B is the size of a message (in bits). $T^c_{t\rightarrow t+1}$ denotes the communication delay to transmit a message from the t-th to $(t+1)$-th server. b_t denotes the bandwidth allocated to the t-th tier server (bits/second).

- *Power Consumption* (f_{PC}): This objective indicates the total power consumption (in Watts) by the CPU cores that operate three VMs in an application.

$$f_{PC} = \sum_{t=1}^{3} \left(P^{q_t}_{idle} + (P^{q_t}_{max} - P^{q_t}_{idle}) \cdot c_t \cdot \frac{q_t}{q_{max}} \right) \tag{7}$$

$P^{q_t}_{idle}$ and $P^{q_t}_{max}$ denote the power consumption of a CPU core that the t-th tier server resides on when its CPU utilization is 0% and 100% at the frequency of q_t, respectively.

AGEGT considers the following four constraints.

- *CPU core capacity constraint* (C_C): The upper limit of the total share allocation on each CPU core. $c_{i,o} \leq C_C$ for all O cores on all M hosts where $c_{i,o}$ is the total share allocation on the o-th core of the i-th host. The violation of this constraint is computed as:

$$g_C = \sum_{i=1}^{M}\sum_{o=1}^{O} \left(I^C_{i,o} \cdot (c_{i,o} - C_C) \right) \tag{8}$$

$I^C_{i,o} = 1$ if $o_i > C_C$. Otherwise, $I^C_{i,o} = 0$.

- *Bandwidth capacity constraint* (C_B): The upper limit of bandwidth consumption allocated to each host. $b_i \leq C_B$ for all M hosts where b_i is the total amount of bandwidth allocated to the i-th host. The violation of this constraint is computed as:

$$g_B = \sum_{i=1}^{M} \left(I^B_i \cdot (b_i - C_B) \right) \tag{9}$$

$I_i^B = 1$ if $b_i > C_B$. Otherwise, $I_i^B = 0$.

- *Response time constraint* (C_{RT}): The upper limit of response time for each application. $f_{RT}^i \leq C_{RT}$ for all N applications where f_{RT}^i is the response time of the i-th application. The violation of this constraint is computed as:

$$g_{RT} = \sum_{i=1}^{N} \left(I_i^{RT} \cdot (f_{RT}^i - C_{RT}) \right) \tag{10}$$

$I_i^{RT} = 1$ if $f_{RT}^i > C_{RT}$. Otherwise, $I_i^{RT} = 0$.

- *Power consumption constraint* (C_{PC}): The upper limit of power consumption for each application. $f_{PC}^i \leq C_{PC}$ for all N applications where f_{PC}^i is the power consumption of the i-th application. The violation of power consumption constraint is computed as:

$$g_{PC} = \sum_{i=1}^{N} \left(I_i^{PC} \cdot (f_{PC}^i - C_{PC}) \right) \tag{11}$$

$I_i^{PC} = 1$ if $f_{PC}^i > C_{PC}$. Otherwise, $I_i^{PC} = 0$.

3. Background: Evolutionary Game Theory

In a conventional game in the game theory, the objective of a rational player is to choose a strategy that maximizes its payoff. In contrast, evolutionary games are played repeatedly by players randomly drawn from a population [20, 34]. This section overviews key elements in evolutionary games: evolutionarily stable strategies (ESS) and replicator dynamics.

3.1. *Evolutionarily Stable Strategies (ESS)*

Suppose all players in the initial population are programmed to play a certain (incumbent) strategy k. Then, let a small population share of players, $x \in (0, 1)$, mutate and play a different (mutant) strategy ℓ. When a player is drawn for a game, the probabilities that its opponent plays the incumbent strategy k and the mutant strategy ℓ are $1-x$ and x, respectively. Thus, the expected payoffs for the player to play k and ℓ are denoted as $U(k, x\ell + (1-x)k)$ and $U(\ell, x\ell + (1-x)k)$, respectively.

Definition 1. A strategy k is said to be evolutionarily stable if, for every strategy $\ell \neq k$, a certain $\bar{x} \in (0, 1)$ exists, such that the inequality

$$U(k,\ x\ell + (1-x)k) > U(\ell,\ x\ell + (1-x)k) \tag{12}$$

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

If the payoff function is linear, Equation 12 derives:

$$(1-x)U(k,k) + xU(k,\ell) > (1-x)U(\ell,k) + xU(\ell,\ell) \tag{13}$$

If x is close to zero, Equation 13 derives either

$$U(k,k) > U(\ell,k) \text{ or } U(k,k) = U(\ell,k) \text{ and } U(k,\ell) > U(\ell,\ell) \tag{14}$$

This indicates that a player associated with the strategy k gains a higher payoff than the ones associated with the other strategies. Therefore, no players can benefit by changing their strategies from k to the others. This means that an ESS is a solution on a Nash equilibrium. An ESS is a strategy that cannot be invaded by any alternative (mutant) strategies that have lower population shares.

3.2. *Replicator Dynamics*

The replicator dynamics describes how population shares associated with different strategies evolve over time [27]. Let $\lambda_k(t) \geq 0$ be the number of players who play the strategy $k \in K$, where K is the set of available strategies. The total population of players is given by $\lambda(t) = \sum_{k=1}^{|K|} \lambda_k(t)$. Let $x_k(t) = \lambda_k(t)/\lambda(t)$ be the population share of players who play k at time t. The population state is defined by $X(t) = [x_1(t), \cdots, x_k(t), \cdots, x_K(t)]$. Given X, the expected payoff of playing k is denoted by $U(k, X)$. The population's average payoff, which is same as the payoff of a player drawn randomly from the population, is denoted by $U(X, X) = \sum_{k=1}^{|K|} x_k \cdot U(k, X)$. In the replicator dynamics, the dynamics of the population share x_k is described as follows. $\dot{x}_k$ is the time derivative of x_k.

$$\dot{x}_k = x_k \cdot [U(k, X) - U(X, X)] \tag{15}$$

This equation states that players increase (or decrease) their population shares when their payoffs are higher (or lower) than the population's average payoff.

Theorem 1. If a strategy k is strictly dominated, then $x_k(t)_{t\to\infty} \to 0$.

A strategy is said to be strictly dominant if its payoff is strictly higher than any opponents. As its population share grows, it dominates the population over time. Conversely, a strategy is said to be strictly dominated if its payoff is lower than that of a strictly dominant strategy. Thus, strictly dominated strategies disappear in the population over time.

There is a close connection between Nash equilibria and the steady states in the replicator dynamics, in which the population shares do not change over time. Since no players want to change their strategies on Nash equilibria, every Nash equilibrium is a steady state in the replicator dynamics. As described in Section 3.1, an ESS is a solution on a Nash equilibrium. Thus, an ESS is a solution at a steady state in

the replicator dynamics. In other words, an ESS is the strictly dominant strategy in the population on a steady state.

AGEGT maintains a population of deployment strategies for each application. In each population, strategies are randomly drawn to play games repeatedly until the population state reaches a steady state. Then, AGEGT identifies a strictly dominant strategy in the population and deploys VMs based on the strategy as an ESS.

4. AGEGT: An Evolutionary Game Theoretic Scheduler for VMs

AGEGT maintains N populations, $\{\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_N\}$, for N applications and performs games among strategies in each population. A strategy s consists of five parameters to indicate the locations of and the resource allocation for three VMs in a particular application:

$$s(a_i) = \bigcup_{t \in 1,2,3} \left(h_{i,t},\ u_{i,t},\ c_{i,t},\ b_{i,t},\ q_{i,t} \right), \quad 1 < i < N \tag{16}$$

a_i denotes the i-th application. $h_{i,t}$ is the ID of a host that a_i's t-th tier VM is placed to. $u_{i,t}$ is the ID of a CPU core that a_i's t-th tier VM resides on in the host $h_{i,t}$. $c_{i,t}$ and $b_{i,t}$ are the CPU and bandwidth allocation for a_i's t-th tier VM. $q_{i,t}$ denotes the frequency of a CPU core that a_i's t-th tier VM resides on.

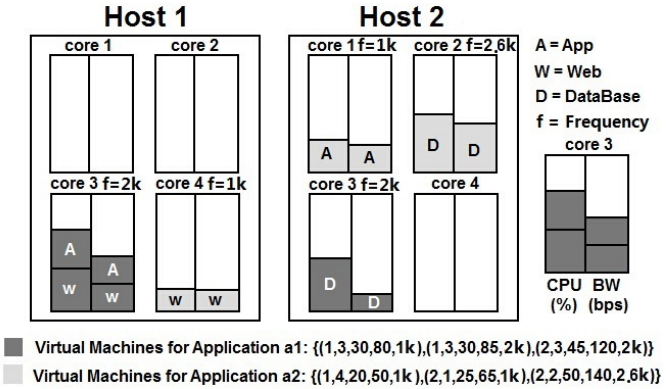


Fig. 2. Example Deployment Strategies

Fig. 2 shows two example strategies for two applications (a_1 and a_2) ($N = 2$). Four cores are available in each of two hosts ($M = 3$ and $O = 2$). a_1's strategy, $s(a_1)$, places the first-tier VM on the third core in the first host ($h_{1,1} = 1$ and $u_{1,1} = 3$). The 30% time share of the CPU core and 80 Kbps bandwidth are allocated to the VM ($c_{1,1} = 30$ and $b_{1,1} = 80$). The VM requires the frequency of 1 GHz for the CPU

core ($q_{1,1} = 1k$). The second-tier VM of a_1 is placed on the third core in the first host ($h_{1,2} = 1$, $u_{1,2} = 3$). 30% of the CPU core time and 85 Kbps bandwidth are allocated to the VM ($c_{1,2} = 30$ and $b_{1,2} = 85$). The VM requires the frequency of 2 GHz for the CPU core ($q_{1,2} = 2k$). The third-tier VM of a_1 requires the frequency of 2 GHz ($q_{1,3} = 2k$) on the third core of the second host ($h_{1,3} = 2$, $u_{1,3} = 3$). 45% of the CPU core time and 120 Kbps bandwidth are allocated to the VM ($c_{1,3} = 45$ and $b_{1,3} = 120$). If multiple VMs are placed on a CPU core, the core operates at the highest required frequency. For example, on the third core of the first host, two VMs requires 1 GHz and 2 GHz. Thus, the core operates at 2 GHz.

Given $s(a_1)$, a_1's objective values for CPU and bandwidth allocation are 105% (30 + 30 + 45) and 285 kbps (80 + 85 + 120). Assuming the CPU core capacity constraint $C_C = 100\%$ (Equation 8), it is satisfied on every core ($g_C = 0$). For example, on the third core of the first host, the total share allocation $c_{1,3}$ is 60% (30% + 30%).

Algorithm 1 Evolutionary Process in AGEGT

$g = 0$
Randomly generate N populations for N applications: $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_N\}$
while $g < G_{max}$ **do**
  for each population $\mathcal{P}_i$ randomly selected from $\mathcal{P}$ **do**
    $\mathcal{P}_i' \leftarrow \emptyset$
    for $j = 1$ to $|\mathcal{P}_i|/2$ **do**
      $s_1 \leftarrow$ randomlySelect($\mathcal{P}_i$)
      $s_2 \leftarrow$ randomlySelect($\mathcal{P}_i$)
      $\{winner, loser\} \leftarrow$ performGame(s_1, s_2)
      $replica \leftarrow$ replicate($winner$)
      $a_{i,t} \leftarrow argmax_{a_{i,t} \in a_i}\ u_t(replica)$
      for each parameter v in $a_{i,t}$ **do**
        if random() $\leq P_m$ **then**
          $replica \leftarrow$ mutate($replica$, v)
        end if
      end for
      $winner' \leftarrow$ performGame($loser$, $replica$)
      $\mathcal{P}_i \setminus \{s_1, s_2\}$
      $\mathcal{P}_i' \cup \{winner, winner'\}$
    end for
    $\mathcal{P}_i \leftarrow \mathcal{P}_i'$
    $d_i \leftarrow argmax_{s \in \mathcal{P}_i} x_s$
    while d_i is infeasible **do**
      $\mathcal{P}_i \setminus \{d_i\}$
      $d_i \leftarrow argmax_{s \in \mathcal{P}_i} x_s$
    end while
    Deploy VMs for the current application based on d_i.
  end for
  $g = g + 1$
end while

Algorithm 1 shows how AGEGT seeks an evolutionarily stable strategy for each application through evolutionary games. In the 0-th generation, strategies are randomly generated for each of N populations $\{\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_N\}$ (Line 2). Those strategies may or may not be *feasible*. Note that a strategy is said to be feasible if it violates none of four constraints described in Section 2. A strategy is said to be *infeasible* if it violates at least one constraint.

In each generation (g), a series of games are carried out on every population (Lines 4 to 26). A single game randomly chooses a pair of strategies (s_1 and s_2) and distinguishes them to the winner and the loser with respect to the objectives described in Section 2 (Lines 7 to 9). A game is carried out based on the superior-inferior relationship between the two strategies and their feasibility (c.f. `performGame()` in Algorithm 1). If a feasible strategy and an infeasible strategy participate in a game, the feasible one always wins over its opponent. If both strategies are feasible, they are compared with the hypervolume (HV) metric [35]. It measures the volume that a given strategy s dominates in the objective space:

$$HV(s) = \Lambda\left(\bigcup\{x' | s \succ x' \succ x_r\}\right) \tag{17}$$

Λ denotes the Lebesgue measure. x_r is the reference point placed in the objective space. The notion of *Pareto dominance* ($\succ$) is defined as follows. A strategy s_1 is said to dominate another strategy s_2 ($s_1 \succ s_2$) if both of the following conditions hold:

- s_1's objective values are superior than, or equal to, s_2's in all objectives.
- s_1's objective values are superior than s_2's in at least one objectives.

A higher hypervolume means that a strategy is more optimal. Given two strategies, the one with a higher hypervolume value wins a game. If both have the same hypervolume value, the winner is randomly selected.

If both strategies are infeasible in a game, they are compared based on their constraint violation. An infeasible strategy s_1 wins a game over another infeasible strategy s_2 if both of the following conditions hold:

- s_1's constraint violation is lower than, or equal to, s_2's in all constraints.
- s_1's constraint violation is lower than s_2's in at least one constraints.

Once a game determines the winner and the loser, the winner replicates itself (Line 10). The replica is altered through *active-guided mutation*, which mutates a strategy with guided local search (GLS) [30] (Lines 11 to 16). Of three VMs of an application that a given strategy represents ($a_{i,t} \in a_i$), this mutation scheme first identifies the VM that yields the worst performance with modified objective functions (Line 11) and then mutates a parameter(s) for the VM (Lines 12 to 16). Eq. 18

is used to evaluate the t-th VM of an application that a strategy s is represents.

$$\vec{u}_t(s) = \frac{\vec{f'_t}(s)}{1 + P_k} \tag{18}$$

$\vec{f'_t}(s) = \{f'_{t,C}(s), f'_{t,B}(s), f'_{t,RT}(s), f'_{t,PC}(s)\}$ is a vector of modified objective functions. $f'_{t,C}(s)$ is computed based on the CPU allocation for the t-th VM of an application that a strategy s represents. Modified objective functions are defined as follows.

$$\vec{f'_t}(s) = \vec{f_t}(s) + \lambda \sum_{k=1}^{U} P_k I_{k,t}(s) \tag{19}$$

$\vec{f_t}(s) = \{f_{t,C}(s), f_{t,B}(s), f_{t,RT}(s), f_{t,PC}(s)\}$. Each function $f_t(s)$ is computed on a VM by VM basis by customizing the original objective function (Eq. 1, 2, 3 or 7). For example, $f_{t,C}(s)$ indicates the CPU allocation for the t-th VM of an application that a strategy s represents. U denotes the total number of CPU cores in a cloud: $U = M * O$. P_k is the *penalty* for the k-th CPU core. It is initialized as 0 in the first generation and repeatedly incremented through generations. $I_k(s)$ is a boolean variable that contains 1 if the k-th CPU core is assigned to the t-th VM of an application that s represents and otherwise 0.

Active-guided mutation evaluates the performance of each VM of an application that a strategy s represents, as $\vec{u}_t(s)$ in Eq. 18, and determines the worst VM by comparing $\vec{u}_t(s)$, $1 < t < 3$ through the notion of Pareto dominance. The worst VM is chosen in Line 11. In Eq. 18, k in P_k indicates the CPU core that the t-th VM resides on, and P_k denotes the total amount of penalty that the CPU core has accumulated in the past generations.

Active-guided mutation also uses $\vec{u}_t(s)$ in Eq. 18 as the utility of penalizing each VM. Once it determines the worst VM in Line 11, it increments the penalty for the CPU core that the VM resides on (P_k in Eq. 18). In Lines 12 to 16, it randomly chooses a parameter (or parameters) of the worst VM identified in Line 11 with a certain mutation rate P_m and alters its/their value(s) at random based on polynomial mutation [6]. (`mutate()` in Line 14 implements polynomial mutation.) Key ideas behind active-guided mutation are to (1) identify the worst-performing VM in each application and alter its deployment strategy in the hope that its performance improves and (2) record inferior VM deployment strategies as penalties through generations and use the penalties to help strategies escape from local optima and improve their performance.

Mutation is followed by a game performed between the loser and the mutated winner (Line 17). This is intended to select the top two of three strategies (the winner, loser and mutated winner). The worst of the three strategies disappears in the population.

Once all strategies play games in the population, AGEGT identifies a feasible strategy whose population share (x_s) is the highest and determines it as a dominant

strategy (d_i) (Lines 22 to 26). AGEGT deploys three VMs for an application in question based on the dominant strategy (Line 27).

Another variant EGT-GLS is studies in this paper shown in Algorithm 2. The main procedure is similar to AGEGT, but the replicate is mutated with polynomial mutation (Lines 10 to 18) [6] instead of guided mutation and also EGT-GLS performs *Guided local search* shown in Algorithm **??** to improve the dominant strategy after a dominant strategy is determined(Line 26). Polynomial mutation randomly chooses a parameter (or parameters) in a given strategy with a certain mutation rate P_m and alters its/their value(s) at random (Lines 12 to 14).

5. Stability Analysis

This section analyzes AGEGT's stability (i.e., reachability to at least one of Nash equilibria) by proving the state of each population converges to an evolutionarily stable equillibrium. The proof consists of three steps: (1) designing a set of differential equations that describe the dynamics of the population state (or strategy distribution), (2) proving an strategy selection process has equilibria and (3) proving the the equilibria are asymptotically stable (or evolutionarily stable) . The proof uses the following terms and variables.

- S denotes the set of available strategies. S^* denotes a set of strategies that appear in the population.
- $X(t) = \{x_1(t), x_2(t), \cdots, x_{|S^*|}(t)\}$ denotes a population state at time t where $x_s(t)$ is the population share of a strategy $s \in S$. $\sum_{s \in S^*}(x_s) = 1$.
- F_s is the fitness of a strategy s. It is a relative value determined in a game against an opponent based on the dominance relationship between them. The winner of a game earns a higher fitness than the loser.
- $p_k^s = x_k \cdot \phi(F_s - F_k)$ denotes the probability that a strategy s is replicated by winning a game against another strategy k. $\phi(F_s - F_k)$ is the probability that the fitness of s is higher than that of k.

The dynamics of the population share of s is described as:

$$\begin{aligned} \dot{x}_s &= \sum_{k \in S^*, k \neq s} \{x_s p_k^s - x_k p_s^k\} \\ &= x_s \sum_{k \in S^*, k \neq s} x_k \{\phi(F_s - F_k) - \phi(F_k - F_s)\} \end{aligned} \tag{20}$$

Note that if s is strictly dominated, $x_s(t)_{t \to \infty} \to 0$.

Theorem 2. The state of a population converges to an equilibrium.

Proof. It is true that different strategies have different fitness values. In other words, only one strategy has the highest fitness among others. Given Theorem 1,

assuming that $F_1 > F_2 > \cdots > F_{|S^*|}$, the population state converges to an equilibrium: $X(t)_{t\to\infty} = \{x_1(t), x_2(t), \cdots, x_{|S^*|}(t)\}_{t\to\infty}$
$= \{1, 0, \cdots, 0\}$. □

Theorem 3. The equilibrium found in Theorem 2 is asymptotically stable.

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

$$\dot{z}_s = z_s[c_{s1}(1 - z_s) + \sum_{i=2, i\neq s}^{|s^*|} z_i \cdot c_{si}], \quad s, k = 2, ..., |S^*| \tag{21}$$

where $c_{sk} \equiv \phi(F_s - F_k) - \phi(F_k - F_s))$ and $Z(t) = \{z_2(t), z_3(t),$
$\cdots, z_{|S^*|}(t)\}$ denotes the corresponding downsized population state. Given Theorem 1, $Z_{t\to\infty} = Z_{eq} = \{0, 0, \cdots, 0\}$ of $(|S^*| - 1)$-dimension.

If all Eigenvalues of Jaccobian matrix of $Z(t)$ has negative real parts, Z_{eq} is asymptotically stable. The Jaccobian matrix J's elements are

$$\begin{aligned} J_{sk} &= \left[\frac{\partial \dot{z}_s}{\partial z_k}\right]_{|Z=Z_{eq}} \\ &= \left[\frac{\partial z_s[c_{s1}(1 - z_s) + \sum_{i=2, i\neq s}^{|S^*|} z_i \cdot c_{si}]}{\partial z_k}\right]_{|Z=Z_{eq}} \\ &\quad for\ s, k = 2, ..., |S^*| \end{aligned} \tag{22}$$

Therefore, J is given as follows, where $c_{21}, c_{31}, \cdots, c_{|S^*|1}$ are J's Eigenvalues.

$$J = \begin{bmatrix} c_{21} & 0 & \cdots & 0 \\ 0 & c_{31} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & c_{|S^*|1} \end{bmatrix} \tag{23}$$

$c_{s1} = -\phi(F_1 - F_s) < 0$ for all s; therefore, $Z_{eq} = \{0, 0, \cdots, 0\}$ is asymptotically stable. □

6. Simulation Evaluation

This section evaluates AGEGT's performance, particularly in its optimality and stability, through simulations.

6.1. *Simulation Configurations*

This paper simulates a cloud data center that consists of 100 hosts in a 10×10 grid topology. The grid topology is chosen based on recent findings on efficient

topology configurations in clouds [13, 14]. This paper also assumes five different types of applications. Table 1 shows the message arrival rate (i.e., the number of incoming messages per second) and message processing time (in second) for each of the five application types. This configuration follows Zipf's law [21, 26]. This paper simulates 40 application instances for each type (200 application instances in total).

Application type	1	2	3	4	5
Message arrival rate (λ in Eq. 6)	110	70	40	20	10
Web server (T_1^p in Eq. 4)	0.02	0.02	0.04	0.04	0.08
App server (T_2^p in Eq. 4)	0.03	0.08	0.04	0.13	0.11
DB server (T_3^p in Eq. 4)	0.05	0.05	0.12	0.08	0.11

Table 1. Message Arrival Rate and Message Processing Time

Each host is simulated to operate an Intel Core2 Quad Q6700 CPU, which is a quad-core CPU that has five frequency and voltage operating points (P-states). Table 2 shows the power consumption at each P-state under the 0% and 100% CPU utilization [12]. This setting is used in Equation 7 to compute power consumption objective values.

P-state	Frequency (q)	P_{idle}^q	P_{max}^q
p1	1.600 GHz	82.70 W	88.77 W
p2	1.867 GHz	82.85 W	92.00 W
p3	2.113 GHz	82.91 W	95.50 W
p4	2.400 GHz	83.10 W	99.45 W
p5	2.670 GHz	83.25 W	103.00 W

Table 2. P-states in Intel Core2 Quad Q6700

Table 3 shows the parameter settings for AGEGT. Mutation rate is set to $1/v$ where v is the number of parameters in a strategy. ($v = 5$ as shown in Eq. 16). Every simulation result is the average with 20 independent simulation runs.

Comparative performance study is carried out for AGEGT and its two variants: EGT-GLS and EGT. Algorithm 2 shows the procedure of EGT-GLS, which is similar to AGEGT. EGT-GLS performs polynomial mutation instead of active-guided mutation and GLS-based local search in each generation (i.e., `localSearch()` in Algorithm 2). The local search operator is designed to improve the performance of the dominance strategy d_i (Algorithm 3). It creates Q mutants of d_i iteratively using GLS and replaces d_i with a mutant if the mutant wins over d_i in a game. Through Q iterations, the local search operator keeps the best mutant discovered so far and mutates it when mutation occurs. Another variant, EGT, performs Algorithm 2

Parameter	Value
Number of hosts (M)	100
Number of cores per CPU/host (O in Eq. 6)	4
Number of applications (N)	200
Number of generations (G_{max} in Algo. 1)	500
Population size ($\|\mathcal{P}_i\|$ in Algo. 1)	100
Penalization rate (λ in Algo. 1)	0.5
Mutation rate (P_m in Algo. 1)	$1/v$
Number of local search iterations (Q in Algo. 3)	20
Reference point for HV computation (x_r in Eq. 17)	f_C=400, f_B=4k, f_{PC}=40k, f_{RT}=400

Table 3. Parameter Settings for AGEGT

with local search disabled.

AGEGT is also compared with NSGA-II, which is a well-known multiobjective evolutionary algorithm [6]. AGEGT and NSGA-II use the same parameter settings shown in Table 3. All other NSGA-II settings are borrowed from [6]. Both AGEGT and NSGA-II are implemented with jMetal [9]. Moreover, AGEGT is evaluated in comparison to well-known heuristics, first-fit and best-fit algorithms (FFA and BFA), which have been widely used for adaptive cloud application deployment [2, 11, 18, 19].

Table 4 shows two different combinations of constraints: no constraints (C_∞) and moderate (C_M). C_M is used unless otherwise noted.

Constraint Combinations	C_C (%)	C_B (Kbps)	C_{PC} (W)	C_{RT} (ms)
C_∞	∞	∞	∞	∞
C_M	100	1,000	400	40

Table 4. Constraint Combinations

6.2. *Simulation Results*

Table 5 examines how a mutation-related parameter, called distribution index (η_m in [6]), impacts the performance of AGEGT, EGT-GLS and EGT. This parameter controls how likely a mutated strategy is similar to its original. (A higher distribution index makes a mutant more similar to its original.) In Table 5, the performance of EGT is evaluated with the hypervolume measure that a set of dominant strategies yield in the 500th generation. The hypervolume metric indicates the union of the volumes that a given set of solutions dominates in the objective space [35]. A higher hypervolume means that a set of solutions is more optimal. As shown in Table 5,

Algorithm 2 Evolutionary Process in EGT-GLS

```
1: g = 0
2: Randomly generate N populations for N applications: P = {P_1, P_2, ..., P_N}
3: while g < G_max do
4:   for each population P_i randomly selected from P do
5:     P_i' ← ∅
6:     for j = 1 to |P_i|/2 do
7:       s_1 ← randomlySelect(P_i)
8:       s_2 ← randomlySelect(P_i)
9:       {winner, loser} ← performGame(s_1, s_2)
10:      replica ← replicate(winner)
11:      for each parameter v in replica do
12:        if random() ≤ P_m then
13:          replica ← mutate(replica, v)
14:        end if
15:      end for
16:      winner' ← performGame(loser, replica)
17:      P_i \ {s_1, s_2}
18:      P_i' ∪ {winner, winner'}
19:    end for
20:    P_i ← P_i'
21:    d_i ← argmax_{s∈P_i} x_s
22:    while d_i is infeasible do
23:      P_i \ {d_i}
24:      d_i ← argmax_{s∈P_i} x_s
25:    end while
26:    d_i ← localSearch(d_i)
27:    Deploy VMs for the current application based on d_i.
28:  end for
29:  g = g + 1
30: end while
```

EGT yields the best performance with the distribution index value of 40. Thus, this parameter setting is used for EGT, EGT-GLS and AGEGT in all successive simulations.

Distribution Index	HV	Distribution Index	HV
30	0.823	35	0.828
40	0.830	45	0.827
50	0.825		

Table 5. Impacts of Distribution Index Values on Hypervolume Performance

Fig. 3 studies how AGEGT, EGT-GLS and EGT evolve their hypervolume through generations. Figs. 4 to 6 show the average, maximum and minimum objective values of AGEGT, EGT-GLS and EGT for two different constraints combina-

Algorithm 3 Guided Local Search (`localSearch()`)

Input: d_i: Dominant strategy to improve
Output: Improved dominant strategy

for $i = 1$ to Q **do**
  for each t-th tier VM in d_i **do**
    $RankedVM[] \leftarrow util_{t(CPU,BW,EN,RT)}(strategy)$
  end for
  $PenalizedVM \leftarrow RankedVM[3]$
  for each parameter v in $PenalizedVM$ **do**
    if random() $\leq P_m$ **then**
      $replica \leftarrow$ mutate(d_i, v)
    end if
  end for
  $d_i \leftarrow$ performGame($replica$, d_i)
end for
return d_i

tion (C_∞ and C_M) in the last generation. These figures demonstrate that AGEGT outperforms EGT-GLS and EGT in both objective values and convergence speed. Active-guided mutation aids AGEGT to gain performance improvement effectively. All three algorithms perform better under constraints. This means that constraint handling works properly in games. All successive simulations use the moderate constraint combination (C_M).

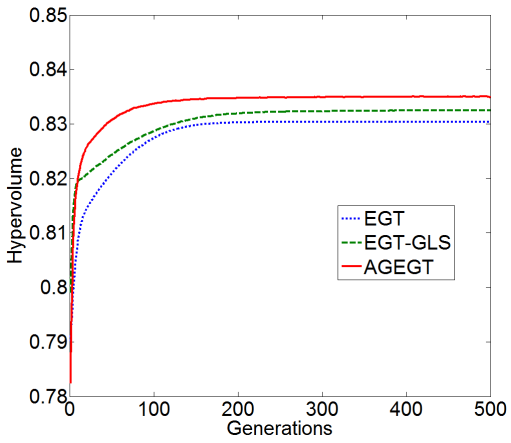
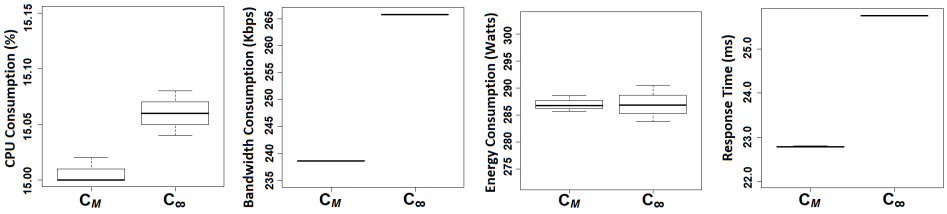


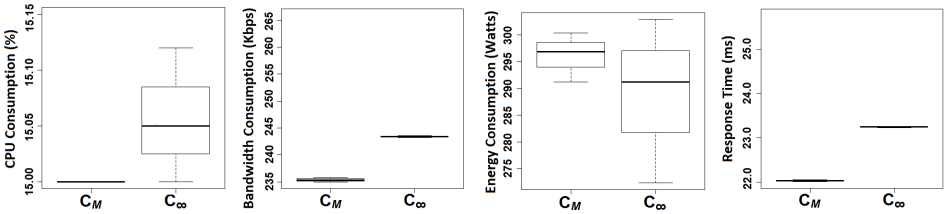
Fig. 3. Comparison of AGEGT, EGT-GLS and EGT in Hypervolume

Table 6 compares AGEGT with NSGA-II, FFA and BFA. It shows the minimum, average and maximum objective values in the last generation. AGEGT outperforms NSGA-II in the average CPU allocation, bandwidth consumption and power consumption by 50.37%, 19.28% and 77.63%, respectively. In response time, NSGA-II outperforms AGEGT by 47.17%. On average, AGEGT outperforms NSGA-II by



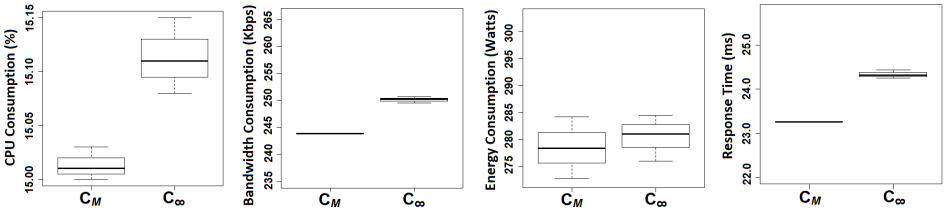
(a) CPU allocation (b) Bandwidth allocation (c) Power consumption (d) Response time

Fig. 4. Objective Values of AGEGT under Two Constraint Combinations



(a) CPU allocation (b) Bandwidth allocation (c) Power consumption (d) Response time

Fig. 5. Objective Values of EGT-GLS under Two Constraint Combinations



(a) CPU allocation (b) Bandwidth allocation (c) Power consumption (d) Response time

Fig. 6. Objective Values of EGT under Two Constraint Combinations

25.02%. FFA and BFA produce two extreme results. FFA yields the lowest power consumption (59.61 Watts) because it is designed to deploy VMs on the minimum number of hosts; however, it sacrifices the other objectives. BFA performs the best in CPU allocation (28.94 %) because it is designed to deploy VMs on the hosts that maintain higher resource availability. AGEGT maintains balanced objective values in between FFA and BFA while performing better in response time, CPU allocation and bandwidth allocation.

Table 8 shows the variance of objective values that AGEGT and NSGA-II yield at the last generation in 20 different simulation runs. A lower variance means higher stability (or higher similarity) in objective value results (i.e., lower oscillations in

Objectives		Min	Avg	Max
CPU allocation (%/app)	AGEGT	15.00	15.05	15.12
	NSGA-II	28.86	30.29	31.35
	FFA	28.68	28.68	28.68
	BFA	28.94	28.94	28.94
Bandwidth allocation (Kbps/app)	AGEGT	238.55	238.65	238.71
	NSGA-II	278.32	288.27	295.89
	FFA	1186	1186	1186
	BFA	1200	1200	1200
Power consumption (W/app)	AGEGT	285.71	286.72	288.60
	NSGA-II	1245.15	1246	1246.92
	FFA	59.12	59.61	60.02
	BFA	341.74	341.85	341.95
Response time (msec/app)	AGEGT	22.77	22.78	22.80
	NSGA-II	11.56	11.79	12.04
	FFA	109.06	109.06	109.06
	BFA	92.09	92.09	92.09

Table 6. Comparison of AGEGT, NSGA-II, FFA and BFA in Objective Values

objective value results) among different simulation runs. AGEGT maintains significantly higher stability than NSGA-II in all objectives except response time. AGEGT's average stability is 98.86% higher than NSGA-II's. This result exhibits AGEGT's stability property (i.e. ability to seek evolutionarily stable strategies), which NSGA-II does not have.

Objectives	AGEGT	NSGA-II	Diff (%)
CPU allocation	0.06	2.16	97.22%
Bandwidth allocation	0.001	5.599	99.98%
Power consumption	0.010	0.747	98.66%
Response time	0.001	0.239	99.58%
Average Difference (%)	–	–	98.86%

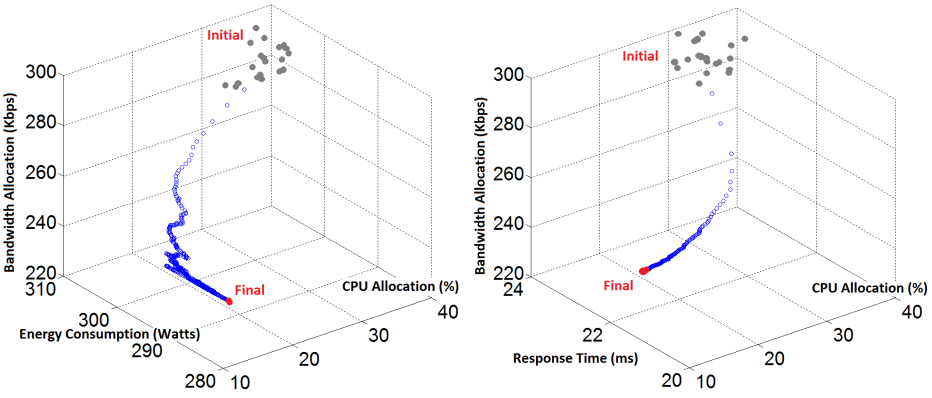
Table 7. Stability of Objective Values in AGEGT and NSGA-II

Fig. 7 shows two three-dimensional objective spaces that plot a set of dominant strategies obtained from individual populations at each generation. Each blue dot indicates the average objective values that dominant strategies yield at a particular generation in 20 simulation runs. The trajectory of blue dots illustrates a path through which AGEGT's strategies evolve and improve objective values. Gray and red dots represent 20 different sets of objective values at the first and last genera-

Objectives	AGEGT	NSGA-II(HV)	NSGA-II(Best)	Max	Avg	Min
CPU allocation	0.15	0.207	0.156	0.45	0.29	0.15
Bandwidth allocation	0.238	0.282	0.15	0.45	0.288	0.15
Power consumption	0.143	0.498	0.497	0.498	0.498	0.496
Response time	0.227	0.11	0.18	0.18	0.11	0.06
Euclidean Distance	0.388	0.758	0.572	0.827	0.653	0.542
Manhattan Distance	0.618	1.097	0.983	1.578	1.186	0.983
Hypervolume	0.835	0.834	0.86	-	0.813	-

Table 8. Distance metric of normalized Objective Values in AGEGT and NSGA-II

tion in 20 simulation runs, respectively. While initial (gray) dots disperse (because strategies are generated at random initially), final (red) dots are overlapped in a small region. Consistent with Table 8, Fig. 7 verifies AGEGT's stability: reachability to at least one Nash equilibria regardless of the initial conditions.



(a) CPU allocation, Bandwidth allocation and Energy consumption

(b) CPU allocation, Bandwidth allocation and Response time

Fig. 7. Trajectory of AGEGT's Solution through Generations

7. Related Work

Numerous research efforts have been made to study heuristic algorithms for application placement problems in clouds (e.g., [2, 4, 11, 17–19, 29, 32]). Most of them assume single-tier application architecture and considers a single objective. For example, in [4,17,29,32], only energy saving is considered as the objective. In contrast, AGEGT assumes a multi-tier application architecture (i.e., three tiers in an application) and considers multiple objectives. It is designed to seek a trade-off solution

among conflicting objectives.

Game theoretic algorithms have been used for a few aspects of cloud computing; e.g., application placement [8, 16, 33], task allocation [24] and data replication [15]. In [8, 16, 33], greedy algorithms seek equilibria in application placement problems. This means they do not attain the stability to reach equilibria as does.

Several genetic algorithms (e.g., [25, 31]) and other stochastic optimization algorithms (e.g., [3, 10]) have been studied to solve application placement problems in clouds. They seek the optimal placement solutions; however, they do not consider stability. In contrast, AGEGT aids applications to seek evolutionarily stable solutions and stay at equilibria.

This paper reports a set of extensions to the authors' prior work [22]. For the problem formulation, this paper considers two extra optimization constraints in addition to the two constraints considered in [22]. As for the algorithmic design, this paper enhances the way to compare two strategies in a game with the hypervolume metric. This paper also investigates active-guided mutation, which is out of the scope of [22]. To the best of the authors' knowledge, this paper is the first attempt to integrate GLS with an evolutionary game theoretic algorithm. For simulation evaluation, this paper conducts more comprehensive comparative study than [22] with extra benchmark algorithms (FFA, BFA and EGT-GLS) and a modern evaluation metric, the hypervolume metric. The algorithm proposed in [22] is basically same as EGT, which this paper uses in its comparative study.

8. Conclusions

This paper proposes and evaluates AGEGT, an evolutionary game theoretic framework for adaptive and stable VM deployment in DVFS-enabled clouds. It theoretically guarantees that every application (i.e., a set of VMs) seeks an evolutionarily stable deployment strategy, which is an equilibrium solution under given workload and resource availability. Simulation results verify that AGEGT performs VM deployment in an adaptive and stable manner. AGEGT outperforms existing well-known heuristics in the quality and stability of VM deployment. For example, AGEGT outperforms NSGA-II by 25% in deployment quality and 98% in deployment stability.

References

[1] 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*, Proc. of acm symposium on operating systems principles, 2003October.

[2] Henri Casanova, Mark Stillwell, and Frédéric Vivien, *Virtual machine resource allocation for service hosting on heterogeneous distributed platforms*, Proc. ieee int'l parallel & distributed processing symposium, 2012.

[3] X. Chang, B. Wang, L. Jiqiang, W. Wang, and K. Muppala, *Green cloud virtual network provisioning based ant colony optimization*, Proc. acm int'l conference on genetic and evol. computat, 2013.

[4] Shuyi Chen, Kaustubh R. Joshi, Matti A. Hiltunen, Richard D. Schlichting, and William H. Sanders, *Blackbox prediction of the impact of DVFS on end-to-end performance of multitier systems*, ACM SIGMETRICS Performance Eval. Rev. **37** (2010), no. 4.

[5] Yi Cheng-Ren, Junichi Suzuki, Athanasios Vasilakos, Shingo Omura, and Katsuya Oba, *Cielo: An evolutionary game theoretic framework for virtual machine placement in clouds*, The 2nd international conference on future internet of things and cloud, 2014.

[6] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, *A fast and elitist multiobjective genetic algorithm: NSGA-II*, IEEE Trans Evol. Computat. **6** (2002), no. 2.

[7] Kalyanmoy Deb, Samir Agrawal, Amrit Pratab, and T. Meyarivan, *A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II*, Proc. conf. parallel problem solving from nature, 2000.

[8] 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 Comm. **30(3)** (2007).

[9] J.J. Durillo, A.J. Nebro, and E. Alba, *The jMetal framework for multi-objective optimization: Design and architecture*, Proc. ieee congress on evol. computat. 2010.

[10] Yongqiang Gao, Haibing Guan, Zhengwei Qi, Yang Hou, and Liang Liu, *A multi-objective ant colony system algorithm for virtual machine placement in cloud computing*, J. Computer and System Sciences **79** (2013), no. 8.

[11] Hadi Goudarzi and Massoud Pedram, *Energy-efficient virtual machine replication and placement in a cloud computing system*, Proc. ieee int'l conf. on cloud comput. 2013.

[12] Tom Guerout, Thierry Monteil, Georges Da Costa, Rodrigo Neves Calheiros, Rajkumar Buyya, and Mihai Alexandru, *Energy-aware simulation with dvfs*, Simulation Modelling Practice and Theory **39** (2013), 96–91.

[13] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, *Bcube: A high performance, server-centric network architecture for modular data centers*, Proc. of acm sigcom, 2009.

[14] C. Guo, H. Wu, K. Tan, L. Shiy, Y. Zhang, and S. Lu, *Dcell: A scalable and fault-tolerant network structure for data centers*, Proc. of acm sigcom, 2008.

[15] S. Khan and I. Ahmad, *A pure Nash equilibrium based game theoretical method for data replication across multiple servers*, IEEE T. Knowl. Data En. **21** (2009), no. 4.

[16] S. U. Khan and C. Ardil, *Energy efficient resource allocation in distributed computing systems*, Proc. of int'l conf. on distrib., high-perf. and grid comp. 2009.

[17] Dzmitry Kliazovich, Pascal Bouvry, and Samee Ullah Khan, *DENS: data center energy-efficient network-aware scheduling*, Cluster Computing (16(1), 2013).

[18] Xin Lia, Zhuzhong Qiana, Sanglu Lua, and Jie Wu, *Energy efficient virtual machine placement algorithm with balanced and improved resource utilization in a data center*, Mathematical and Computer Modelling (June 58(5).

[19] Fei Ma, Feng Liu, and Zhen Liu, *Multi-objective optimization for initial virtual machine placement in cloud data center*, J. Infor. and Computational Science **9** (2012), no. 16.

[20] M. A. Nowak, *Evolutionary dynamics: Exploring the equations of life*, Harvard University Press, 2006.

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

[22] Y. Ren, J. Suzuki, C. Lee, A. V. Vasilakos, S. Omura, and K. Oba, *Balancing performance, resource efficiency and energy efficiency for virtual machine deployment in DVFS-enabled clouds: An evolutionary game theoretic approach*, Proc. of acm genetic and evolutionary computation conference, 2014.

[23] T. C. Shan and W. W. Hua, *Solution architecture for n-tier applications*, Proc. of ieee int'l conference on services computing, 2006September.

[24] R. Subrata, A. Y. Zomaya, and B. Landfeldt, *Game theoretic approach for load balancing in computational grids*, IEEE Trans. Parall. Distr. **19** (2008), no. 1.

[25] H. A. Taboada, J. F. Espiritu, and D. W. Coit, *MOMS-GA: A Multi-Objective Multi-State Genetic Algorithm for System Reliability Optimization Design Problems*, IEEE Trans. Reliability **57** (2008), no. 1.

[26] J. Tatemura, W-P. Hsiung, and W-S. Li, *Acceleration of web service workflow execution through edge computing*, Proc. of int'l www conference, 2003.

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

[28] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, *An analytical model for multi-tier internet services and its applications*, Proc. of acm int'l conference on measurement and modeling of computer systems, 2005June.

[29] von Laszewski, Lizhe Wang, Andrew J. Younge, and Xi He, *Power-aware scheduling of virtual machines in DVFS-enabled clusters*, Proc. ieee int'l conf. on clusters, 2009.

[30] C. Voudouris and E.P.K. Tsang, *Guided local search and its application to the travelling salesman problem* **113** (1999), no. 2, 469–499.

[31] H. Wada, J. Suzuki, Y. Yamano, and K. Oba, *E3: A multiobjective optimization framework for sla-aware service composition*, IEEE Trans. Services Computing **5** (2012), no. 3.

[32] Qingyang Wang, Yasuhiko Kanemasa, Jack Li, Chien An Lai, Masazumi Matsubara, and Calton Pu, *Impact of DVFS on n-tier application performance*, Proc. acm conference on timely results in operating systems, 2010.

[33] G. Wei, A. V. Vasilakos, Y. Zheng, and N. Xiong, *A game-theoretic method of fair resource allocation for cloud computing services*, J. Supercomputing **54** (2009), no. 2.

[34] J. W. Weibull, *Evolutionary game theory*, MIT Press, 1996.

[35] E. Zitzler and L. Thiele, *Multiobjective optimization using evolutionary algorithms: A comparative study*, Proc. int'l conf. on parallel problem solving from nature, 1998.