Exploring Self-Optimization and Self-Stabilization Properties in Bio-inspired Autonomic Cloud Applications

Paskorn Champrasert\textsuperscript{1} Junichi Suzuki\textsuperscript{2,*} and Chonho Lee\textsuperscript{3}

\textsuperscript{1} Chiang Mai University, Thailand
\textsuperscript{2} University of Massachusetts, Boston, USA
\textsuperscript{3} Nanyang Technological University, Singapore

SUMMARY

This paper describes an architecture to build self-optimizable and self-stabilizable cloud applications. The design of the proposed architecture, SymbioticSphere, is inspired by key biological principles such as decentralization, evolution and symbiosis. In SymbioticSphere, each cloud application consists of application services and middleware platforms. Each service and platform is designed as a biological entity and implements biological behaviors such as energy exchange, migration, reproduction and death. Each service/platform possesses behavior policies, as genes, each of which governs when to and how to invoke a particular behavior. SymbioticSphere allows services and platforms to autonomously adapt to dynamic network conditions by optimizing their behavior policies with a multiobjective genetic algorithm. Moreover, SymbioticSphere allows services and platforms to autonomously seek stable adaptation decisions as equilibria (or symbiosis) between them with a game theoretic algorithm. This symbiosis augments evolutionary optimization to expedite the adaptation of agents and platforms. It also contributes to stable performance that contains a very limited amount of fluctuations. Simulation results demonstrate that agents and platforms successfully attain self-optimization and self-stabilization properties in their adaptation process.

KEY WORDS: Cloud computing; bio-inspired autonomic networking; genetic algorithms; game theory; optimization; stabilization

\textsuperscript{*}Correspondence to: Department of Computer Science, University of Massachusetts, Boston, Boston, MA 02125, USA. E-mail: jxs@cs.umb.edu

Received 31 January 2000

Copyright © 2000 John Wiley & Sons, Ltd.

Revised 19 September 2002
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, 4, 5]. In order to provide this feature, cloud computing environments (or simply clouds) are required to dynamically adjust each application’s configurations such as location, resource utilization and availability [6, 7, 8, 9]. For example, clouds are required to allocate different amounts of computing and networking resources (e.g., CPU cores, memory space and bandwidth) to each application according to its workload (i.e., the number of incoming user requests). This allows applications to operate by balancing their performance (e.g., response time) and operational costs. (Operational costs depend on resource consumption due to the pay-per-use models used in clouds.) Moreover, clouds relocate an application from one host to another, or 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 operating applications in clouds:

- **Self-optimization**: allows applications to autonomously seek their optimal configurations (e.g., locations, resource utilization and availability), as adaptation decisions, under dynamic network conditions.
- **Self-stabilization**: allows applications to autonomously seek stable adaptation decisions by minimizing oscillations (or non-deterministic inconsistencies) in decision making.

SymbioticSphere is an architecture to build self-optimizable and self-stabilizable cloud applications. Its design is inspired by key biological principles and mechanisms based on an observation that various biological systems have developed the mechanisms necessary to attain self-optimization and self-stabilization properties.

In SymbioticSphere, each app is constructed with two types of components: application services and middleware platforms. Each of them is modeled as a biological entity, analogous to an individual bee in a bee colony. They are designed to follow several biological principles such as decentralization, emergence, evolution and symbiosis. An application service is designed as a software agent. It implements a functional service (e.g., web service) and biological behaviors such as energy exchange, reproduction, replication, migration and death. A middleware platform provides runtime services that agents use to perform their services and behaviors, and implements biological behaviors such as energy exchange, reproduction and death. SymbioticSphere models agents and platforms as different biological species.

In SymbioticSphere, each agent and platform autonomously senses its surrounding network conditions and adaptively invokes a behavior suitable for the conditions. For example, an agent may invoke the migration behavior to move toward a host that receives a large number of user requests for its services. This results in the adaptation of agent location; the agent can improve its response time for users. Also, a platform may invoke the reproduction behavior to make its offspring on a neighboring host where resource availability is high. This results in the adaptation of resource availability; the platforms provide more resources to agents.

Each agent/platform possesses behavior policies, each of which defines when to and how to invoke a particular behavior. A behavior policy is encoded as a set of genes. SymbioticSphere
implements a genetic algorithm (GA) that allows agents and platforms to evolve behavior policies (i.e., genes) via genetic operations such as selection, mutation and crossover when agents/platforms replicate themselves or reproduce their offspring. This evolution process allows agents/platforms to evolve and adapt their behavior policies to dynamic network conditions across generations. Evolution also frees application developers from anticipating all possible network conditions and optimizing their agents and platforms to the conditions at design time. Instead, agents and platforms can evolve and autonomously optimize themselves to network conditions. This can significantly simplify the implementation and maintenance of agents/platforms.

In addition to the aforementioned (regular) behaviors, agents and platforms perform a special type of behaviors: symbiotic behaviors. A symbiotic behavior is a sequence of regular behaviors that an agent and its underlying platform invoke in order. As described above, agents and platforms can adapt to dynamic network conditions by invoking regular behaviors; however, regular behaviors of one species (e.g., agents) can degrade the adaptation of the other species (e.g., platforms). For example, if too many agents migrate to a host for reducing response time to user request, a platform on the host has a risk to crash due to overloading or resource scarcity. Symbiotic behaviors are intended for agents and platforms to balance and augment their adaptability by allowing the two species to pursue their mutual benefits.

SymbioticSphere employs a game theoretic approach to implement symbiotic behaviors. It allows each agent and its underlying platform to play a game and find an equilibrium where they can agree with a symbiotic behavior as a rational sequence of regular behaviors. It is theoretically proven that the two players always reach an equilibrium regardless of their internal states (e.g., history of behavior invocation in the past) and external states (e.g., surrounding network conditions). Thanks to this stability property, SymbioticSphere guarantees that agents and platforms deterministically performs symbiotic behaviors in an adaptive and stable manner.

A key novelty in SymbioticSphere is that it integrates optimization and stabilization processes and allows agents/platforms to dynamically balance the optimality and stability in their adaptation. Optimization and stabilization have been studied largely in isolation. Traditional GAs and other stochastic search algorithms often focus on optimization and fail to seek stable solutions [10, 11]. As a result, they can inconsistently yield different sets of optimal or semi-optimal solutions in different runs/trials with the same problem settings. Conversely, traditional game theoretic algorithms are often dedicated to seek equillibrium solutions, which are not necessarily optimal [12].

This paper describes the genetic and game theoretic algorithms in SymbioticSphere and evaluates their impacts on the self-optimization and self-stabilization properties in cloud applications. Simulation results show that agents and platforms autonomously adapt to dynamic network conditions (e.g., network traffic and resource availability) by optimizing their behavior policies through evolution. Agents and platforms yield stable performance results that contain a very limited amount of fluctuations.
2. Design Principles in SymbioticSphere

SymbioticSphere applies the following biological principles to design agents and platforms.

**Decentralization:** In various biological systems (e.g., bee colony), there are no central entities to control or coordinate individual entities in order to increase scalability and survivability. Similarly, in SymbioticSphere, there are no central entities to control and coordinate agents and platforms.

**Autonomy:** Inspired by biological entities (e.g., bees), agents/platforms sense their surrounding network conditions, and based on the conditions, they behave and interact with each other without any intervention from/to other agents, platforms and human users.

**Emergence:** In biological systems, collective (group) behaviors emerge from interactions of individual entities. In SymbioticSphere, agents/platforms interact only with nearby peers. Desirable system characteristics (e.g., adaptability) emerge through collective behaviors and interactions of individual agents/platforms.

**Lifecycle and Food Chain:** Biological entities strive to seek and consume food for living. In SymbioticSphere, agents/platforms store and expend energy for living. Each agent gains energy in exchange for performing its service to other agents or human users, and expends energy to use computing and network resources (Figure 1). Each platform gains energy in exchange for providing resources to agents, and periodically evaporates energy (Figure 1). The abundance or scarcity of stored energy in agents/platforms affects their lifecycle. For example, an abundance of stored energy indicates high demand to an agent/platform; thus, the agent/platform favors reproduction or replication to increase its availability. A scarcity of stored energy indicates a lack of demand; it causes death of the agent/platform.

In the ecosystem, the energy accumulated from food is transferred between different species to balance their populations. For example, producers (e.g., shrubs) convert the Sun light energy to chemical energy (Figure 1). The chemical energy is transferred to consumers (e.g., hares) as consumers consume producers [14]. In SymbioticSphere, the energy exchange among users, agents and platforms is designed after ecological food chain (Figure 1). SymbioticSphere models a user as the Sun, agents as producers, and platforms as consumers. Similar to the Sun, each user has an unlimited amount of energy. When a user requests a service implemented by an agent, the user transfers a certain amount of energy to the agent. (Each agent specifies the price of its service in energy units.) Each agent gains energy from users and transfers 10% of its energy level to the underlying platform for consuming resources provided by the platform. Each platform gains energy from agents and periodically evaporates 10% of its energy level to the environment. This energy exchange rule follows an ecological fact that a consumer species acquires approximately 10% of the energy that a producer species maintains [14].

**Evolution:** Biological entities evolve as a species so that the entities that fit better to the environment become more abundant [15]. In SymbioticSphere, agents and platforms evolve by generating behavioral diversity and executing selection. Behavioral diversity means that different agents/platforms possess different behavior policies. This is generated via mutation and crossover during replication and reproduction. Selection is triggered with agents’ and platforms’ energy levels. It retains the agents/platforms whose energy levels are high (e.g., the agents that have effective behavior policies, such as moving toward a user to gain more energy) and eliminates the agents/platforms whose energy levels are low (e.g., the agents that
have ineffective behavior policies, such as moving too often). Through successive generations, effective behavior policies become abundant in an agent/platform populations while ineffective ones become dormant or extinct. This evolutionary process allows agents/platforms to adapt to dynamic network conditions.

**Symbiosis**: Competition for food and terrain always occurs in the biological world; however, several species establish mutual relationships to avoid excessive competition and support with each other to survive [16]. In SymbioticSphere, agents and platforms are designed as different species and pursue their mutual benefits (i.e., gaining more energy to survive). Each agent and its underlying platform play an extensive-form game to find an equilibrium as the most rational sequence of behaviors and perform it as a symbiotic behavior.

### 3. Background: Extensive-form Games

This section overviews extensive-form games, which SymbioticSphere implements for agents and platforms to seek and agree on symbiotic behaviors as equilibria. An extensive-form game is a particular specification of a game in game theory [13]. This form describes each game as a tree. See Figure 2 for an example extensive-form game. Each non-terminal node represents a player, and the player invokes a certain behavior at that node. The behavior choice is represented as an edge leading from that node to another node in a lower tier. In Figure 2, two players (A and B) participate in a game. $a_i^j$ denotes a behavior choice in which player $i$
invokes behavior \( j \). Thus, player \( A \) has three behavior choices \((a_1^A, a_2^A, a_3^A)\), and player \( B \) has two behavior choices \((a_1^B, a_2^B)\).

An extensive-form game begins at the root node and flows through a path (i.e., a set of edges) depending on the behavior choices that players make. A game ends when it reaches a terminal node, and payoffs are assigned to all players. In Figure 2, player \( A \) invokes a behavior first because it is assigned to the root node. Player \( B \) observes \( A \)'s behavior choice and then invokes a behavior. There are six potential outcomes represented by six terminal nodes after \( A \) and \( B \) invoke one behavior each. \( A \)'s payoff is denoted by \( p_i^A \) where \( i \) indicates a player and \( S \) indicates a sequence of behaviors. If \( A \) invokes \( a_1^A \) and \( B \) invokes \( a_1^B \), a game ends by reaching the top terminal node. In this case, \( A \)'s payoff is \( p_{(a_1^A,a_1^B)}^A \), and \( B \)'s payoff is \( p_{(a_1^A,a_1^B)}^B \).

Unlike a normal-form game, an extensive-form game models a sequential interaction between players. If all players have chosen behaviors and no players can gain higher payoffs by changing their behaviors while the other players keep their behaviors unchanged, the current sequence of behaviors and its corresponding payoffs constitute a Nash equilibrium. It has been theoretically proven that there exist at least one Nash equilibrium in an extensive-form game [13].

In an extensive-form game, each player attempts to gain as high payoff as possible by looking ahead which behaviors the other players invoke. An extensive-form game can be solved with backward induction in order to obtain an Nash equilibrium [13]. In Figure 2, Player \( B \) considers which behavior \((a_1^B \text{ or } a_2^B)\) it should invoke to gain higher payoff \((\max(p_{(a_1^B,a_1^A)}^B, p_{(a_1^B,a_2^A)}^B), \max(p_{(a_2^B,a_1^A)}^B, p_{(a_2^B,a_2^A)}^B))\) in the cases that Player \( A \) invokes \( a_1^A, a_2^A \) and \( a_3^A \). If \( p_{(a_1^A,a_1^B)}^B > p_{(a_1^B,a_1^A)}^B \), \( p_{(a_2^B,a_1^A)}^B < p_{(a_2^B,a_2^A)}^B \) and \( p_{(a_3^B,a_1^A)}^B > p_{(a_3^B,a_3^A)}^B \), Player \( B \) invokes \( a_1^B, a_2^B \) and \( a_3^B \) in the cases that Player \( A \) invokes \( a_1^A, a_2^A \) and \( a_3^A \). (See three thick edges from Node \( B \) to terminal nodes in Figure 2.) Given the knowledge of Player \( B \)'s behavior choices (i.e., those thick edges), Player \( A \) considers which behavior \((a_1^A, a_2^A \text{ or } a_3^A)\) it should invoke to gain as high payoff as possible: \( \max(p_{(a_1^A,a_1^B)}^A, p_{(a_1^A,a_2^B)}^A, p_{(a_1^A,a_3^B)}^A) \). If \( p_{(a_1^A,a_1^B)}^A \) is the highest payoff, Player \( A \) invokes \( a_1^A \). (See a thick edge between Node \( A \) and Node \( B \) in Figure 2.)
As a result, a Nash equilibrium solution is determined as \((a_A^3, a_B^1)\). This backward induction guarantees to find at least one Nash equilibrium.

4. SymbioticSphere

Each agent runs on a platform. A platform is an execution environment (or middleware) for agents. It abstracts low-level operating and networking details (e.g., network I/O), and implements high-level runtime services that agents use to perform their services and behaviors. Each platform can operate multiple agents, and each host operates at most one platform.

4.1. Agents

Each agent consists of attributes, body and behaviors. Attributes carry descriptive information on an agent, such as its energy level, description of a service it provides, and price (in energy units) of the service it provides. Body implements a service that an agent provides. For example, an agent may implement a web service, while another may implement a physical model for scientific simulations. Behaviors implement actions that are inherent to all agents:

- **Replication**: Agents may make a copy of themselves. A replicated (child) agent is placed on the platform that its parent agent resides on, and it inherits the half amount of the parent’s energy level.
- **Reproduction**: Agents may produce their offspring with their mating partners. A child agent is placed on the platform that its parent agent resides on, and it receives the half amount of the parent’s energy level.
- **Death**: Agents die due to energy starvation. When an agent dies, its underlying platform removes the agent and releases all resources allocated to the agent.
- **Migration**: Agents may move from one platform to another.

4.2. Platforms

Each platform consists of attributes, behaviors and runtime services. Attributes carry descriptive information on the platform, such as its energy level and health level. Health level is defined as a function of three properties: the resource availability on, the age of and the freshness of an underlying host. Resource availability indicates how much resources are available for agents and platforms on a host. Age indicates how long a host has been alive. Freshness indicates how recently a host joined the network. Once a host joins the network, its freshness gradually decreases from the maximum. When an unstable host resumes from a failure, its freshness starts with the value that the host had when it went down.

Health level affects how platforms and agents invoke their behaviors. For example, higher health level indicates longer uptime of and/or higher resource availability on a host that a platform resides on. Thus, the platform may replicate itself on a neighboring host if the host is healthier than the local host. This results in the adaptation of platform locations. Platforms strive to concentrate around long-lived and resource-rich hosts. Also, lower health
level indicates that a platform runs on a host that is short-lived and/or poor in resources. Thus, agents may leave the platform and migrate to a healthier neighboring hosts. This results in the adaptation of agent locations. Agents strive to concentrate around stable and/or resource-rich hosts. In this case, the platforms on short-lived and/or resource-poor hosts will eventually die due to energy starvation because agents do not run on the platforms and pay energy to them. This results in the adaptation of platform population. Platforms avoid running on the hosts that are short-lived and/or poor in resources.

Behaviors are the actions inherent to all platforms:

- **Replication**: Platforms may make a copy of themselves. A replicated (child) platform is placed on a neighboring host that does not run a platform. (Since there is only one type of platform, two or more platforms are not allowed to run on each host.) It inherits the half of the parent’s energy level.

- **Reproduction**: Platforms may produce their offspring with their mating partners. A child platform is placed on a neighboring host that does not run a platform. It inherits the half of the parent’s energy level.

- **Death**: Platforms die due to lack of energy. A dying platform kills agents running on it, uninstalls itself and releases all resources the platform uses. Despite the death of a platform, its underlying host remains active so that another platform can run on it in the future.

**Runtime services** are the middleware services that agents and platforms use to perform their behaviors.

### 4.3. Behavior Policies

Agents and platforms invoke two types of behaviors: regular behaviors and symbiotic behaviors. Sections 4.1 and 4.2 described regular behaviors. Figure 3 shows how agents and platforms make decisions on behavior invocation. Each agent/platform possesses policies for its behaviors. A behavior policy governs when to and how to invoke a particular behavior.

```plaintext
1  While (not simulation last cycle)
2      For each agent Do
3          decide which symbiotic behavior to invoke.
4          if no symbiotic behavior is invoked
5              decide which regular behavior to invoke.
6      End For
7      For each platform Do
8          decide which symbiotic behavior to invoke.
9          if no symbiotic behavior is invoked
10             decide which regular behavior to invoke.
11             update health level.
12             evaporate energy.
13      End For
14  End While
```

**Figure 3. Invocation of Symbiotic and Regular Behaviors**

**Figure 4. Agent-Platform Game**
4.3.1. Symbiotic Behavior Policies

Symbiotic behavior policies are the behavior policies that agents and platforms have for their symbiotic behaviors. Each of them seeks an equilibrium sequence of regular behaviors with an extensive-form game (Section 3) and determines it as a symbiotic behavior for an agent and its underlying platform.

Figure 4 shows an example extensive-form game of an agent (A) and its underlying platform (P). \(a_i^j\); \(i\) denotes a behavior choice where player \(i\) (A or P) invokes behavior \(j\) (\(R\): reproduction, \(M\): migration or \(D\): do nothing). A can invoke \(a_A^R\), \(a_A^M\) or \(a_A^D\). P can invoke \(a_P^R\) or \(a_P^D\). In each extensive-form game, A invokes a behavior first at the root node. P observes A’s behavior choice then invokes its own behavior. A payoff is denoted as \(p_i^S\) where \(i\) and \(S\) indicate a player (A or P) and a sequence of behaviors, respectively. For example, if A chooses \(a_A^M\) and P chooses \(a_P^R\), A’s payoff is \(p_A^{A_M,A_P^R}\) and P’s payoff is \(p_P^{B_M,A_P^R}\).

A payoff is computed based on a combination of network condition benefits. Equation 1 represents how to calculate a network condition benefit (\(\Delta\)). \(\rho\) represent a specific network condition (i.e., the resource availability, the number of hop counts from an agent to a user access point). The network condition benefit (\(\Delta\)) is calculated as the difference of the network condition before (\(\rho_t\)) and the network condition after (\(\rho_{t+1}\)) Agent A and Platform P invoke symbiotic behavior. Positive \(\Delta\) value represents that the network condition is improved after the symbiotic behavior is invoked. Negative \(\Delta\) value represents that the network condition is degraded after the symbiotic behavior is invoked.

\[
\Delta = \rho_{t+1} - \rho_t
\]  

The network condition benefits that an agent consider for its payoff are:

- **Hop Counts:** The number of hop counts from an agent to a user access point, which encourages agents to move towards a user.
- **Resource Availability:** The resource availability of of the underlying platform, which encourages agents to work on resource rich platform.
- **Workload:** The workload per agent, which encourages agents to work on high service request platform.

The network condition benefits that a platform consider for its payoff are:

- **Resource Availability:** The resource availability that the platform can provides to agents, which encourages platforms to increase resource availability for agents.
- **The number of agents:** The number of agents working on the platform, which encourages platforms to gain high energy from agents working on it.

To compare two payoffs, a domination ranking mechanism [17] is applied.

4.3.2. Regular Behavior Policies

Regular behavior policies are the behavior policies that each agent/platform has for its regular behaviors. Each policy consists of factors (\(F_i\)), which evaluate network conditions (e.g., network
traffic) or agent/platform status (e.g., energy/health level). Each factor is given a weight ($W_i$). A behavior is invoked if the weighted sum of corresponding factor values ($\sum F_i \times W_i$) exceeds a threshold.

The factors for the \textit{agent migration behavior} are:

- \textit{Energy Level}: Agent energy level, which encourages agents to move in response to their high energy level.
- \textit{Health Level Ratio}: The ratio of health level on a neighboring platform to the local platform, which encourages agents to move to healthier platforms. This ratio is calculated with three health level properties. Equation 2 represents how to calculate the health level ratio ($HR$). $HLPN_i$ and $HLPL_i$ represent the health level which includes resource availability ($i=1$), freshness ($i=2$) and age ($i=3$) on neighboring platform/host and on local platform/host respectively.

$$HR = \sum_{i=1}^{3} \frac{HLPN_i - HLPL_i}{HLPL_i}$$

- \textit{Service Request Ratio}: The ratio of the number of incoming service requests on a neighboring platform to the local platform. This factor encourages agents to move toward users.
- \textit{Migration Interval}: Time interval to perform migration, which discourages agents to migrate too often.

If there are multiple neighboring platforms that an agent can migrate to, the agent calculates the weighted sum of the above factors for each neighboring platform, and moves to a platform that generates the highest sum.

The factors for the \textit{agent reproduction/replication behavior} are:

- \textit{Energy Level}: Agent energy level, which encourages agents to reproduce their offspring in response to their high energy levels.
- \textit{Request Queue Length}: The length of a queue, which the local platform stores incoming service requests. This factor encourages agents to reproduce their offspring in response to high demands for their services.

When the weighted sum of the above factor values exceeds a threshold, an agent seeks a mating partner from the local and neighboring platforms. If a mating partner is found, the agent invokes the reproduction behavior. Otherwise, the agent invokes the replication behavior. Section 4.4 describes how an agent seeks its mating partner.

The factors for the \textit{agent death behavior} are:

- \textit{Energy Level}: Agent energy level. Agents die when they run out of their energy.
- \textit{Energy Loss Rate}: The rate of energy loss ($ER$), calculated with Equation 3. $E_i$ and $E_{i-1}$ denote the energy levels in the current and previous time instants. Agents die in response
to sharp drops in demands for their services.

\[ ER = \frac{E_{t-1} - E_t}{E_{t-1}} \]  

(3)

The factors for the **the platform reproduction/replication behavior** are:

- **Energy Level**: Platform energy level, which encourages platforms to reproduce their offspring in response to their high energy levels.
- **Health Level Ratio**: The ratio of health level on a neighboring host to the local host. This factor encourages platforms to reproduce their offspring on the hosts that generate higher values with Equation 2.
- **The Number of Agents**: The number of agents working on each platform. This factor encourages platforms to reproduce their offspring in response to high agent population on them.

When the weighted sum of the above factor values exceeds a threshold, a platform seeks a mating partner from its neighboring hosts. If a mating partner is found, the platform invokes the reproduction behavior. Otherwise, it invokes the replication behavior. Section 4.4 describes how a platform finds its mating partner. If there are multiple neighboring hosts that a platform can place its child platform on, it places the child on a host whose health level ratio is highest among others.

The factors for the **the platform death behavior** are:

- **The Number of Agents**: The number of agents running on each platform. This factor discourages platforms to die when agents run on them.
- **Energy Loss Rate**: The rate of energy loss, calculated with Equation 3. Platforms die in response to sharp drops in demands for their resources.

Each agent/platform expends energy to invoke behaviors (i.e., behavior cost) except the death behavior. When the energy level of an agent/platform exceeds the cost of a behavior, it decides whether it performs the behavior by calculating a weighted sum described above.

### 4.4. Evolutionary Process

The weight and threshold values in behavior policies have significant impacts on the adaptability of agents and platforms. However, it is hard to anticipate all possible network conditions and find an appropriate set of weight and threshold values for the conditions. As shown in Section 4.3.2 there are 18 weight and threshold values in regular behaviors. Assuming that 10 different values can be assigned to each weight and threshold, there are \(10^{18}\) possible combinations of weight and threshold values.

Instead that data center designers manually assign weight and threshold values, SymbioticSphere allows agents and platforms to autonomously find appropriate values in an evolutionary manner, thereby adapting themselves to network conditions. Both regular and
symbiotic behavior policies are encoded as genes of agents and platforms. Each gene contains one or more weight values and a threshold value for a particular behavior.

For regular behaviors, each agent/platform has a gene (i.e., a set of weight and threshold values) for each behavior. Figures 5(a) and 5(b) show the gene structure for agent/platform behaviors. For example, for the agent reproduction behavior, a gene is structured to have three elements: (1) $W_{a1}$, a weight value for the energy level factor; (2) $W_{a2}$, a weight value for the factor of request queue length; and (3) $Tr^a$, a threshold value (Figure 5(a)).

Each weight value is a decimal number in the range of [0,1], and it is initialized randomly. Each threshold value is a decimal number in the range of [0...M], where M denotes the number of considered factors. Each threshold value is also initialized randomly. For example, the agent reproduction behavior has a threshold value in the range of [0...2] because the behavior considers two factors: $W_{r1}$ and $W_{r2}$ (see Section 4.3.2 and Figure 5(a)).

The genes of agents and platforms are altered via genetic operations (genetic crossover and mutation) when they perform the reproduction and replication behaviors. As described in Sections 4.1 and 4.2, each agent/platform selects a mating partner when it performs the reproduction behavior. A mating partner is selected by ranking agents/platforms running on the local and neighboring hosts. For this ranking process, SymbioticSphere uses a domination ranking mechanism [17].

Agents and platforms are ranked with two objectives: 1) energy level, 2) the number of behavior invocations. The energy level indicates the total energy that an agent/platform has. The number of behavior invocations indicates how many times an agent/platform invokes a behavior. In all objectives, the higher, the better.

For instance, agents/platforms are plotted on a two dimensional space whose axes are the objectives. Figure 6 shows an example to rank four different agents (Agent A to D). In this example, Agent A dominates the other three agents in both of two objectives. (In other words, Agent A is non-dominated.) Therefore, the agent A is given Rank 1. Agent B is dominated by Agent A; however, it dominates the other two agents (Agent C and D). Thus, Agent B is given Rank 2. Agent C and D are dominated by Agent B, and they cannot dominate with each other. Thus, they are given Rank 3.

During reproduction, an agent/platform ranks other agents/platforms running on the local and neighboring hosts, as described above, and selects the one in the highest rank as a mating partner. If the parent agent/platform, which invokes the reproduction behavior, is in the highest rank, it fails to find its mating partner and performs the replication behavior.
In reproduction, a genetic crossover occurs. A parent and its mating partner contribute their genes and randomly combine them for a child’s gene (Figure 7). Then, a genetic mutation occurs on the child’s gene. Each gene element (i.e., weight or threshold value) is randomly altered with a mutation probability (Figure 7).

In replication, a parent copies its gene to its child. Then, a mutation occurs on the child’s gene in the same way as the mutation in reproduction.

5. Simulation Results

This section shows a set of simulation results to evaluate the self-optimization and self-stabilization properties in adaptation of agents and platforms. Simulations were conducted with the SymbioticSphere simulator†, which implements the mechanisms described in Section 4.

5.1. Simulation Configurations

This section describes a series of simulation configurations. Figure 8(a) shows a simulated data center. It consists of 25 hosts in a grid topology. The grid topology is chosen based on recent findings on efficient topology configurations in data centers [18, 19]. Each agent implements a web service. At the beginning of each simulation, one agent and one platform are deployed on each host. Each agent/platform is initialized with randomly-generated behavior policies.

†The current code base of the SymbioticSphere simulator contains 16,400 lines of Java code. This simulator is freely available at http://dssg.cs.umb.edu/projects/SymbioticSphere
Users send service requests to agents via user access point (Figure 8(a)). This paper assumes that a single (virtual) user runs on the access point, and it emulates multiple users to send out service requests. Figure 8(b) shows how the (virtual) user changes service request rate over time. It is obtained from the workload trace of the 1998 Soccer World Cup official website (June 21-27, 1998) [20]. The peak workload is 2,500 requests/second.

This simulation study assumes a heterogeneous data center. 80% of hosts (20 hosts) have 1 GB memory space, and 20% of hosts (five hosts) have 2 GB memory space. (The two types of hosts are placed randomly in a data center.) Of the total memory space on a host, an operating system consumes 192 MB. The remaining space is available for a platform and agents on each host. Each agent and platform consumes 5 and 20 MB, respectively. This assumption is obtained from a prior empirical experiment [21].

Each host operates in the active or inactive state. If a platform works on a host, the host is active and consumes at least 149W power when there is no workload on the host. An active host with 100% CPU utilization consumes 258W. Power consumption increases linearly as CPU utilization increases. A host becomes inactive when a platform dies on it. An inactive host consumes 2W. These assumptions on power consumption are obtained from [22, 23]. A host is assumed to become active from the inactive state using the Wake On LAN (WOL) technology [24]. When a platform replicates itself onto an inactive host, the platform sends a WOL packet to the host to wake it up.

This simulation study evaluates two configurations of SymbioticSphere:

- **SymbioticSphere-GA (SS-GA)**: Agents and platforms evolve their regular behavior policies and invoke regular behaviors. They do not carry symbiotic behavior policies and

---

[20] Each host is assumed to be Dell Desktop Dimension XPS400 with a Pentium4 CPU (http://www.dell.com/).
do not invoke symbiotic behaviors. They use the mutation probability of 5%. Uniform crossover is performed when they invoke the reproduction behavior.

- **SymbioticSphere-GAGT (SS-GAGT):** Agents and platforms carry both symbiotic and regular behavior policies and invoke both types of behaviors. They use the same configurations for genetic operations as in SS-GA.

### 5.2. Invocation of Symbiotic Behaviors

This section evaluates how agents and platforms invoke symbiotic behaviors in SS-GAGT. Figure 9(a) shows the service request rate at the fifth day (June 26) of the 1998 Soccer World Cup. (It is an excerpt from the 120th to the 144th hour in Figure 8(b).) Figure 9(b) shows the number of symbiotic behavior invocations in SS-GAGT under the workload depicted in Figure 9(a). A black bar in Figure 9(b) represents the number of symbiotic behavior invocations in which agents perform the migration behavior and underlying platforms perform the reproduction behavior ($M-R$). A gray bar represents the number of symbiotic behavior invocations in which agents perform the reproduction behavior and underlying platforms perform the reproduction behavior ($R-R$).

As Figure 9(b) shows, the number of symbiotic behavior invocations stays low when service request rate is low (e.g., in the first nine hours). Agents and platforms invoke the symbiotic behavior $M-R$ when service request rate is relatively low. In this symbiotic behavior, platforms reproduce their offspring onto nearby inactive hosts so that agents can migrate to those hosts in order to distribute workload among hosts.

When service request rate spikes at 9am, the number of symbiotic behavior invocations dramatically increases. Particularly, agents and platforms often invoke the symbiotic behavior $R-R$ to effectively process incoming service requests. In this symbiotic behavior, agents reproduce their offspring on their local hosts and platforms reproduce their offspring onto nearby inactive hosts so that both agents and platforms can increase their availability rapidly. This allows reproduced agents can migrate to reproduced platforms to process a higher workload with more resources, thereby improving throughput. Figure 9(b) demonstrates that agents and platforms effectively use their symbiotic behavior policies (Section 4.3.1) and successfully invoke their symbiotic behaviors against dynamic network conditions.

### 5.3. Adaptability

This section evaluates how agents and platforms self-optimize their configurations (e.g., locations, resource allocation and availability) for adapting to dynamic network conditions. Figures 10 to 13 show the average and range values in each evaluation metric. They are obtained through 12 independent simulations. An average value is depicted as a circle dot. A range value represents the average maximum-minimum difference, shown as an error bar.

Figures 10(a) and 10(b) show the average response time for agents to process one service request from the user in SS-GA and SS-GAGT, respectively. The response time includes the request transmission latency between the user and an agent and the processing overhead for an agent to process a service request. Figure 10(a) shows that, in SS-GA, agents and platforms gradually self-optimize their configurations (e.g., locations, resource allocation and availability)
and improve response time performance by evolving their regular behavior policies. Response time stays high in the beginning of a simulation because the initial regular behavior policies of agents and platforms are randomly-generated.

Figure 10(b) shows that agents and platforms improve response time more quickly in SS-GAGT than in SS-GA. This demonstrates that they effectively invoke symbiotic behaviors in addition to evolution of regular behavior policies. Moreover, agents and platforms yield smaller fluctuation and variance in response time in SS-GAGA than in SS-GA. This demonstrates that symbiotic behaviors contribute to stable response time performance as equilibrium adaptation decisions between agents and platforms.

Figures 11(a) and 11(b) show the average throughput in SS-GA and SS-GAGT, respectively. Throughput is measured as the ratio of the number of service requests agents process to the total number of service requests the user issues. Figure 11(a) illustrates that, in SS-GA, agents and platforms gradually self-optimize their configurations and improve throughput performance by evolving their regular behavior policies.

Figure 11(b) shows that agents and platforms expedite their adaptation in throughput by invoking both regular and symbiotic behaviors. In SS-GAGT, they maintain the throughput of 98% or higher nearly always during a simulation. In contrast, SS-GA fails to consistently maintain high throughput when service request rate spikes. For example, throughput drops to 62% in Day 3 because agents and platforms have not evolved their regular behavior policies yet at that time. Symbiotic behaviors contribute to stable throughput performance as equilibrium adaptation decisions between agents and platforms.

Figures 12(a) and 12(b) show the average resource efficiency in SS-GA and SS-GAGT, respectively. Resource efficiency is measured with Equation 4 where $S$ denotes the total number of service requests that agents process in the entire data center and $R$ denotes the total amount
Figure 10. Response Time

Figure 11. Throughput
of memory space that agents and platforms consume in the entire data center. Higher resource efficiency means that agents and platforms process more service requests with less resources.

\[
\text{Resource efficiency} = \frac{S}{R}
\]  

(4)

In both SS-GA and SS-GAGT, resource efficiency follows the changes in service request rate shown in Figure 8(b). Given the throughput performance in Figure 11, this means that agents and platforms self-optimize their availability and in turn resource utilization as service request rate changes. (Note that \(S\) in Equation 4 indicates throughput.)

SS-GA and SS-GAGT yield qualitatively similar resource efficiency; however, SS-GA’s resource efficiency is often a little higher than SS-GAGT’s particularly in the first two days. This is because (1) SS-GA’s throughput is lower than SS-GTGA’s (Figure 11) and (2) SS-GTGA encourages agents and platforms to cooperate with symbiotic behaviors to survive longer, thereby maintaining their higher availability than in SS-GA. Higher availability means higher resource utilization; i.e., higher \(R\) in Equation 4.

Figures 13(a) and 13(b) show how resource utilization is distributed over active hosts in SS-GA and SS-GAGT, respectively. It is measured as resource utilization balancing index (RUBI) with Equation 5. \(R_i\) denotes the resource utilization on the (active) host \(i\) (i.e., the ratio of the amount of memory space a platform and agents utilize on the host \(i\) to the total amount of resources the host \(i\) has). \(\mu\) denotes the average of \(R_i\) among active hosts. \(N\) denotes the number of active hosts. Lower RUBI indicates higher distribution of resource utilization among active hosts.

\[
\text{Resource Utilization Balancing Index} = \sqrt{\frac{\sum_{i=1}^{N} (R_i - \mu)^2}{N}}
\]  

(5)
Compared with SS-GA (Figure 13(a)), SS-GAGT yields significantly lower RUBI (Figure 13(b)). This indicates that agents and platforms better consider resource availability on hosts and effectively operate on resource-rich hosts as often as possible. Moreover, agents and platforms yield smaller fluctuation and variance in RUBI in SS-GAGA than in SS-GA. This demonstrates that symbiotic behaviors contribute to stable RUBI performance as equilibrium adaptation decisions between agents and platforms.

5.4. Stability

This section evaluates how agents and platforms self-stabilize their performance. Four performance metrics are used: response time, throughput, resource efficiency and RUBI. Table I shows the average and fluctuation of performance metric values over time. The average is measured as a mean of hour-by-hour values in a performance metric during a simulation run. The fluctuation is measured as a standard deviation of hour-by-hour values in a performance metric during a simulation run.

For example, the average response time ($\mu^R$) is measured with Equation 6 where $t$ denotes the simulation tick (from the 1st to the 168th hour), $T_i$ denotes the average response time among agents at the $t$-th simulation tick, and $N (=168)$ denotes the duration of a simulation (i.e., total number of ticks in a simulation). Response time fluctuation ($\sigma^T$) is measured with Equation 7.

\[
\mu^T = \sum_{i=1}^{N} \frac{T_i}{N} \tag{6}
\]
Table I. The Average and Fluctuation of Performance Metric Values in a Simulation Run.

<table>
<thead>
<tr>
<th>Performance Metric</th>
<th>SS-GA</th>
<th>SS-GAGT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response Time</td>
<td>4.23 (5.10)</td>
<td>0.86* (0.31*)</td>
</tr>
<tr>
<td>Throughput</td>
<td>94.16 (8.01)</td>
<td>99.57* (2.76*)</td>
</tr>
<tr>
<td>Resource Efficiency</td>
<td>153.82* (96.65)</td>
<td>105.73 (89.20*)</td>
</tr>
<tr>
<td>RUBI</td>
<td>0.36 (0.12)</td>
<td>0.01* (0.05*)</td>
</tr>
</tbody>
</table>

In Table I, a value within parentheses indicates a standard deviation. A better performance metric value between SS-GA and SS-GAGT is marked in a bold font face. A lower fluctuation value between them is also marked in bold. The * symbol indicates that values between SS-GA and SS-GAGT are different at the significance level of 99%. Average values are compared with a t-test. Standard deviation values are compared with Levene’s test equality of variances [25].

Table I illustrates that SS-GAGT yields significantly lower fluctuation than SS-GA in all of four performance metrics. Symbiotic behaviors allow agents and platforms to self-stabilize their performance under dynamic network conditions by maintaining a small amount of performance fluctuation over time. Moreover, SS-GAGT significantly outperforms SS-GA in average performance metrics except resource efficiency. It yields a lower average resource efficiency because it encourages agents and platforms to survive longer via symbiotic behaviors as discussed in Section 5.3. (See Figure 12 as well.)

Table II examines how the hour-by-hour average performance measure fluctuates across different simulation runs. It is computed for the response time metric with Equation 8. $T^\text{max}_t$ and $T^\text{min}_t$ denote the maximum and minimum average response time values at the $t$-th simulation tick among different simulation runs. $\hat{\mu}_T$ denotes the average range between $T^\text{max}_t$ and $T^\text{min}_t$ from the 1st to the N-th (i.e., the 168th) simulation tick. This maximum-minimum range’s standard deviation is computed with Equation 9 and shown as a value within parentheses in Table II. $\hat{\sigma}_T$ values between SS-GA and SS-GAGT are compared with a t-test with the significance level of 99%.

$$\sigma_T = \sqrt{\frac{\sum_{t=0}^{N} (T_t - \mu_T)^2}{N}}$$  \hspace{1cm} (7)

$$\hat{\mu}_T = \frac{\sum_{t=1}^{N} (T^\text{max}_t - T^\text{min}_t)}{N}$$  \hspace{1cm} (8)

$$\hat{\sigma}_T = \sqrt{\frac{\sum_{t=1}^{N} ((T^\text{max}_t - T^\text{min}_t) - \hat{\mu}_T)^2}{N}}$$  \hspace{1cm} (9)

Table II illustrates that SS-GAGT yields significantly smaller fluctuation than SS-GA in all performance metrics across different simulation runs. The standard deviation of this performance fluctuation is smaller in SS-GAGT in all metrics. Symbiotic behaviors allow
5.5. Power Efficiency

This section evaluates how much electric power SS-GA and SS-GAGT consume in a simulation. In this simulation study, SS-GA and SS-GAGT are compared with a traditional data center in which all hosts are always on. This data center performs a load balancing mechanism so that an equal workload is given to each host.

Table III shows SS-GAGT consumes 624 kW, which is 11.73% less than a traditional data center’s power consumption (707 kW). This indicates that SS-GAGT effectively reduces power consumption by deactivating hosts when platforms die atop them. As described in Section 5.1, inactive hosts consume less power than active hosts.

Table III also shows that SS-GAGT consumes more power than SS-GA because SS-GAGT encourages agents and platforms to survive longer via symbiotic behaviors and strives to maintain higher availability of them than SS-GA as discussed in Section 5.3. However, the difference in power consumption is very small between SS-GAGT and SS-GA (0.71%).

5.6. Scalability

This section evaluates the performance of SS-GA and SS-GAGT in a larger-scale data center and discusses their scalability against data center size. This larger-scale data center operates 400 hosts in a 20x20 topology. (It is 16 times larger in terms of the number of hosts than the 25-host data center simulated for the previous sections.) 80% of the hosts (320 hosts) have 2 GB memory space, and 20% of them (80 hosts) have 4 GB memory space. In this data center, each agent and platform are assumed to consume 10 and 40 MB, respectively. All the other simulation configurations are same as the ones described in Section 5.1.
Table IV. The Average and Fluctuation of Performance Metric Values in a Simulation Run (400 Nodes)

<table>
<thead>
<tr>
<th>Performance Metric</th>
<th>SS-GA</th>
<th>SS-GAGT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response Time</td>
<td>14.25 (17.52)</td>
<td><strong>2.92</strong> (1.19*)</td>
</tr>
<tr>
<td>Throughput</td>
<td>95.63 (8.17)</td>
<td><strong>99.54</strong> (2.79*)</td>
</tr>
<tr>
<td>Resource Efficiency</td>
<td><strong>157.82</strong> (96.81)</td>
<td>101.11 (75.20*)</td>
</tr>
<tr>
<td>RUBI</td>
<td>1.36 (0.53)</td>
<td><strong>0.09</strong> (0.41*)</td>
</tr>
</tbody>
</table>

Table V. Performance Fluctuation across Different Simulation Runs (400 Hosts)

<table>
<thead>
<tr>
<th>Performance Metric</th>
<th>SS-GA</th>
<th>SS-GAGT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response Time</td>
<td>13.99 (2.01)</td>
<td><strong>3.86</strong> (0.42)</td>
</tr>
<tr>
<td>Throughput</td>
<td>23.31 (5.99)</td>
<td><strong>8.89</strong> (1.71)</td>
</tr>
<tr>
<td>Resource Efficiency</td>
<td>45.03 (9.66)</td>
<td><strong>18.21</strong> (1.67)</td>
</tr>
<tr>
<td>RUBI</td>
<td>0.33 (0.07)</td>
<td><strong>0.03</strong> (0.03)</td>
</tr>
</tbody>
</table>

Table IV shows the average and fluctuation of performance metric values (i.e., response time, throughput, resource efficiency and RUBI) over time in the 400-host data center. The average and fluctuation are measured for response time with Equations 6 and 7, respectively. SS-GAGT yields significantly lower fluctuation than SS-GA in all performance metrics. SS-GAGT also significantly outperforms SS-GA in average performance metrics except resource efficiency. These results are consistent with the ones in Table I. Tables I and IV demonstrate that the self-optimization and self-stabilization properties of SS-GA and SS-GAGT scale well as the number of hosts increases in a data center.

Table V evaluates how the maximum-minimum performance range fluctuates in SS-GA and SS-GAGT across different simulation runs. It is computed for the response time metric with Equation 8. The performance range’s standard deviation is computed with Equation 9 for response time. Tables II and V demonstrate that SS-GA and SS-GAGT yield similar performance in the 25-host and 400-host data centers. SS-GAGT yields significantly smaller performance range fluctuation than SS-GA in all performance metrics across different simulation runs. The standard deviation of this performance range fluctuation is smaller in SS-GAGT in all metrics. In comparison with Table II, Table V shows that the self-stabilization property of SS-GA and SS-GAGT scales well as the number of hosts increases in a data center.

6. Related Work

This paper describes several extensions to the authors’ prior work [26, 27]. This paper investigates symbiotic behaviors as well as regular behaviors, while only regular behaviors are studied in [26]. Symbiotic behaviors are studied in [27]. However, in [27], each symbiotic behavior is designed as a statically pre-defined sequence of regular behaviors. Agents
and platforms agree on and invoke symbiotic behaviors with a coevolutionary genetic algorithm (GA). In this paper, each agent and its underlying platform dynamically seek an equilibrium sequence of regular behaviors with a game theoretic algorithm and invoke it as a symbiotic behavior. Moreover, this paper investigates self-optimization and self-stabilization properties in SymbioticSphere, while only self-optimization is studied in [26, 27].

SymbioticSphere is similar to Bio-Networking Architecture (BNA) [28], Jack-in-the-Net (Ja-Net) [29], BEYOND [30] and iNet-EGT [31] in that they study similar bio-inspired and decentralized agents for cloud-like applications in dynamic network environments. However, none of them consider platforms as biological entities; they do not address resource efficiency, power efficiency and resource utilization balancing of applications. They also do not consider symbiosis between agents and platforms. BNA implements a GA to seek the optimal behavior policies of agents. However, unlike SymbioticSphere, the stability in agent/platform adaptation is out of BNA’s scope. iNet-EGT leverages an evolutionary game theoretic algorithm for agents to seek evolutionarily stable behavior policies. It does not consider the optimality in adaptation of agents/platforms as SymbioticSphere does.

Organic Grid [32] and So-Grid [33] use bio-inspired agents to build adaptive grid/cloud applications in a decentralized manner, as SymbioticSphere does. Organic Grid addresses task allocation, and So-Grid focuses on resource discovery and load balancing. SymbioticSphere addresses not only these types of adaptation but also adaptation in throughput, resource efficiency, power efficiency and resource utilization balancing. Both Organic Grid and So-Grid do not seek the optimality and stability in adaptation.

[34, 35, 36, 37, 38] study the optimal placement of cloud/grid applications; however, it is out of their scope to seek equilibrium solutions. In SymbioticSphere, agents and platforms seek equilibria as symbiotic behaviors and pursue the optimal behavior policies through evolution. They possess stability in finding equilibria.

GAs have been used for adaptation of grid/cloud applications (e.g., [28, 39, 40, 41, 42, 43]). All of these efforts seek the optimal adaptation strategies; however, they do not consider stability in adaptation. Moreover, each of them uses a fitness function to rank genes/agents (or adaptation strategies). It aggregates multiple optimization objectives as a weighted sum. Application developers need to manually configure those weight values through trial and errors. In contrast, no parameters exist for ranking agents/platforms in SymbioticSphere because of its domination ranking mechanism. SymbioticSphere incurs much less configuration tasks/costs.

Game theoretic algorithms have been used in several aspects of cloud/grid applications; for example, task allocation [44], application placement [8, 45, 46, 47] and data replication [48]. [44] maintains stability in seeking equilibria; however, it assumes static networks whose conditions (e.g., network traffic) never change over time. [8, 45, 46] formulate equilibria in application placement and use greedy algorithms to seek equilibrium solutions. Thus, they fail to attain stability in seeking equilibria. In contrast, SymbioticSphere maintains stability in finding equilibrium solutions (i.e., symbiotic behaviors) in dynamic networks. [47, 48] is similar to SymbioticSphere in that it maintains stability to seek equilibria. However, it does not consider both optimality and stability in adaptation of applications, while SymbioticSphere does.

A limited number of research efforts consider optimization and stabilization simultaneously (e.g., [49, 50, 51]). [49] seeks optimality and stability in adaptation decisions in static networks, while SymbioticSphere maintains them in dynamic networks as well. Similar to
SymbioticSphere, [50, 51] integrate game theoretic models with GAs for making routing decisions in ad-hoc networks. Their payoff functions have several parameters that need to be manually configured (e.g., threshold and time frame size). In contrast, SymbioticSphere’s payoff functions and fitness function have no parameters to manually configure.

7. Conclusion

This paper describes and evaluates an application architecture, called SymbioticSphere, to build self-optimizable and self-stabilizable cloud applications. Through its bio-inspired mechanisms, SymbioticSphere allows cloud applications (services and middleware platforms) to autonomously adapt to dynamic network conditions by optimizing their behavior policies with a genetic algorithm. It also allows services and platforms to autonomously seek stable adaptation decisions as equilibria between them and yield stable performance results that contain a very limited amount of fluctuations.

Several extensions are planned on SymbioticSphere as future work. First, the SymbioticSphere simulator will be extended to accommodate the notion of virtualization for operating data centers. Currently, it simulates a collection of hosts rather than a collection of virtual machines (VMs) atop CPU cores. Future simulation studies will evaluate SymbioticSphere with VM-based cloud applications. Second, the GA in SymbioticSphere will be compared with other optimization algorithms such as ant colony optimization algorithms and particle swarm optimization algorithms.

REFERENCES


