Self-Stabilizable Symbiosis for Cloud Data Center Applications: A Game Theoretic Perspective

Junichi Suzuki
University of Massachusetts, Boston
Department of Computer Science
Boston, MA 02125, U.S.A.
jxs@cs.umb.edu

Paskorn Champrasert
Chiang Mai University
Department of Computer Engineering
Chiang Mai, 50200, Thailand
paskorn@eng.cmu.ac.th

Chonho Lee
Nanyang Technological University
Department of Computer Engineering
Nanyang Ave., 639798, Singapore
leechonho@ntu.edu.sg

***Abstract*—This paper describes and evaluates a self-stabilizable adaptation framework for data center applications. The design of the proposed architecture, SymbioticSphere, is inspired by key biological principles such as decentralization, natural selection, emergence and symbiosis. In SymbioticSphere, each data center application consists of application services and middleware platforms. Each service and platform is designed as a biological entity, analogous to an individual bee in a bee colony, and implements biological behaviors such as energy exchange, migration, replication and death. SymbioticSphere allows services and platforms to (1) adaptively invoke their behaviors according to dynamic network conditions and (2) autonomously seek stable behavior invocations as equilibria (or symbiosis) between them. A symbiosis between a service and a platform is sought as a Nash equilibrium in an extensive-form game. Simulation results demonstrate that SymbioticSphere allows services and platforms to successfully adapt to dynamic networks in a self-stabilizable manner.**

***Index Terms*—Adaptive networking, self-stabilization, Cloud data center, game theory, extensive-form games**

I. Introduction

One of key features in cloud data centers is *elastic scaling* of their applications [1]. In order to provide this feature, cloud data centers are required to dynamically adjust each application's configurations such as location, resource utilization and availability [2], [3].

This paper investigates two properties in elastic scaling:

- *Self-adaptation*: allows applications to autonomously adapt their configurations to dynamic network conditions (e.g., workload and resource availability).
- *Self-stabilization*: allows applications to autonomously seek stable adaptation decisions by avoiding non-deterministic inconsistencies in decision making.

SymbioticSphere is an architecture to build self-adaptive and self-stabilizable cloud applications. It is designed after key biological principles and mechanisms based on an observation that various biological systems have attained autonomy, adaptability and stabilizability.

In SymbioticSphere, each application 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, natural selection 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, 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, replication and death.

SymbioticSphere allows services and platforms to (1) adaptively invoke their behaviors according to dynamic network conditions and (2) autonomously seek stable behavior invocations as equilibria (or symbiosis) between them. A symbiosis between a service and a platform is sought as a Nash equilibrium in an extensive-form game. Simulation results demonstrate that services and platforms successfully adapt to dynamic networks in a self-stabilizable manner.

II. SymbioticSphere

In SymbioticSphere, agents run on platforms, which in turn run on network hosts. Each platform can operate multiple agents, and at most one platform runs on a host.

A. Design Principles

SymbioticSphere leverages the following bio-inspired principles to design agents and platforms.

Decentralization: There are no central entities (e.g., directories and resource managers) to control and coordinate agents/platforms. Decentralization is intended to improve the scalability and survivability of agents/platforms by avoiding a single point of performance bottlenecks and failures.

Autonomy and emergence: Agents and platforms periodically sense their local 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. 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. Moreover, a platform may invoke the replication 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. Through collective behavior invocations and interactions of

agents and platforms, desirable system characteristics such as adaptability emerge in a swarm of agents and platforms. Note that those desirable characteristics are not present in any single agent/platform.

Energy exchange and natural selection: Agents and 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 network and computing 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 triggers *selection* of agents/platforms. For example, an abundance of stored energy indicates higher demand for an agent/platform; thus the agent/platform replicates itself to increase its availability. A scarcity of stored energy (an indication of lack of demand) causes death of the agent/platform. Like in biological natural selection where more favorable species in an environment becomes more abundant, the population of agents/platforms dynamically changes based on the demands for them.

Symbiosis: Agents and platforms are modeled as different species. They spontaneously cooperate, in certain circumstances, to balance and augment their adaptability by allowing the two species to pursue their mutual benefits (i.e., gaining more energy to survive longer). To this end, they perform a special type of behaviors: *symbiotic behaviors*. A symbiotic behavior is a sequence of regular behaviors (e.g., migration and replication) that an agent and its underlying platform invoke in order.

As described above, agents and platforms are designed to adapt to dynamic network conditions by invoking their (regular) behaviors. However, behavior invocations of one species (e.g., agents) can degrade the adaptation of the other species (e.g., platforms) in some circumstances. For example, if too many agents migrate to a host near a user for gaining more energy from the user and reducing response time to user requests, 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 invoke behaviors as coalitional decisions in a cooperative manner rather than individual decisions in a selfish manner.

B. 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.

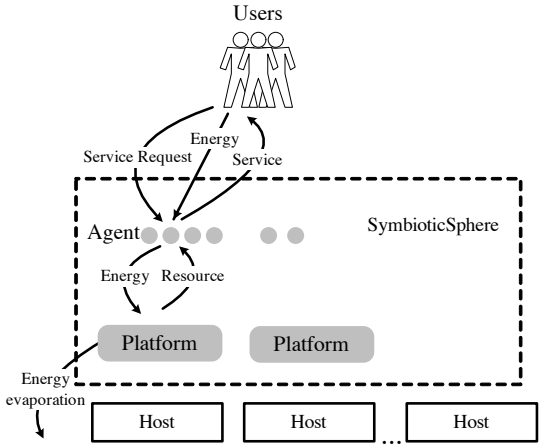

Fig. 1: Energy Exchange in SymbioticSphere

- **Migration**: Agents may move from one platform to another.
- **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.

C. 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 a 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 (i.e., how much stable the host is). Freshness indicates how recently a host joined the network. Once a host joins the network, its freshness gradually decreases from the maximum. When a 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 favor 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 favor 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 few agents run on the platforms and transfer energy to them. This results in the adaptation of platform population. Platforms strive to 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. 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

D. Agent-Platform Symbiosis

There are two types of symbiotic behaviors: (1) *agent-initiated* symbiotic behaviors, each of which is a sequence of an agent's behavior and its underlying platform's behavior, and (2) *platform-initiated* symbiotic behaviors, each of which is a sequence of a platform's behavior and its local agent's behavior.

SymbioticSphere employs a game theoretic approach to implement symbiotic behaviors. Each agent and its underlying platform play an *extensive-form game* to find an equilibrium where they can agree on (i.e., a rational sequence of regular behaviors) and invoke it as a symbiotic behavior.

An extensive-form game is a particular specification of games in game theory [4]. This form describes each game as a tree. See Figure 2 for the structure of an example extensive-form game for agent-initiated symbiosis between an agent (A) and its underlying platform (P). Each non-terminal node represents a player (A or P), and the player chooses 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. a^i_j denotes a behavior choice in which player i (A or P) invokes behavior j (R: replication, M: migration or D: do nothing). Thus, A has three behavior choices (a^A_R, a^A_M and a^A_D), and B has two behavior choices (a^P_R and a^P_D).

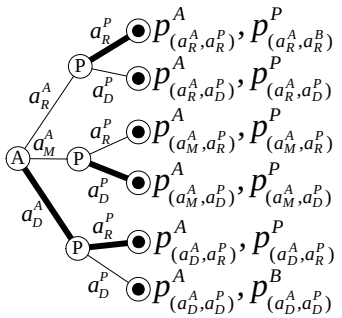

Fig. 2: Agent-Plaform Game for Agent-initiated Symbiosis

An extensive-form game begins at the root node and flows through a path (i.e., a set of edges) depending on the behaviors that players choose. A game ends when it reaches a terminal node, and payoffs are assigned to players. In Figure 2, A chooses a behavior first at the root node. (Note that Figure 2 depicts agent-initiated symbiosis.) P observes A's behavior choice and decides its own behavior. There are six potential outcomes represented by six terminal nodes after A and B choose one behavior each. A payoff is denoted as p^i_S where S denotes a sequence of behaviors. In Figure 2, if A chooses a^A_M and P chooses a^P_R, A's payoff is $p^A_{(a^A_M,a^P_R)}$ and P's payoff is $p^B_{(a^A_M,a^B_R)}$.

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 is theoretically proven that there exist at least one Nash equilibria in an extensive-form game [4]

Equation 1 shows how to compute a payoff Δ). ρ represents a particular network condition. Δ is computed as the difference between the current network condition (ρ_t) and an estimated network condition that a symbiotic behavior yields (ρ_{t+1}). A positive Δ value indicates that a symbiotic behavior is estimated to improve a network condition. A negative Δ value indicates that a symbiotic behavior is estimated to degrade a network condition.

$$\Delta = \rho_{t+1} - \rho_t \tag{1}$$

Three kinds of network conditions are considered to compute an agent's payoff.

- *Hop count*: The number of network hop counts between an agent and the users who request the agent's service.
- *Resource availability*: The amount of resources that the underlying platform made available.
- *Workload*: The workload (i.e., the number of service requests) dispatched to an agent.

Two kinds of network conditions are considered to compute an platform's payoff.

- *Resource availability*: The amount of resources that a platform provides to agents.
- *The number of agents*: The number of agents that runs on a platform.

In order to compare two payoffs and determine which one is superior, SymbioticSphere uses a dominance ranking mechanism that considers the Pareto optimality [5].

SymbioticSphere performs backward induction to obtain a Nash equilibrium in a game between an agent and platform. In backward induction, each player attempts to maximize its payoff by looking ahead which behaviors the other players invoke. In a game in Figure 2, a platform (P) first considers which behavior (a^P_R or a^P_D) it should choose to maximize its payoff: $\max(p^P_{(a^A_R,a^P_R)}, p^P_{(a^A_R,a^P_D)})$, $\max(p^P_{(a^A_M,a^P_R)}, p^P_{(a^A_M,a^P_D)})$, and $\max(p^P_{(a^A_D,a^P_R)}, p^P_{(a^A_D,a^P_D)})$ if an agent (A) chooses a^A_R, a^A_M and a^A_D, respectively. The platform P chooses a^P_R if A chooses a^A_R and $p^P_{(a^A_R,a^P_R)} > p^P_{(a^A_R,a^P_D)}$. This behavior choice is depicted as a thick edge from the top P node to the top terminal node in Figure 2. Similarly, P chooses a^P_D if A chooses a^A_M and

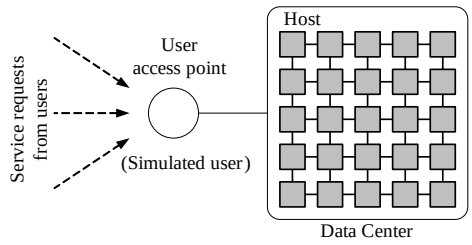

Fig. 3: Simulated Data Center

$p^P_{(a^A_M,a^P_R)} < p^P_{(a^A_M,a^P_D)}$. It chooses a^P_R if A chooses a^A_D and $p^P_{(a^A_D,a^P_R)} > p^P_{(a^A_D,a^P_D)}$.

Given the knowledge of P's behavior choices (i.e., thick edges in Figure 2), A considers which behavior (a^A_R, a^A_M or a^A_D) it should choose to maximize its payoff: $\max(p^A_{(a^A_R,a^P_R)}, p^A_{(a^A_M,a^P_D)}, p^A_{(a^A_D,a^P_R)})$. If $p^A_{(a^A_D,a^P_R)}$ is highest among the three payoffs, A chooses a^A_D. This behavior choice is shown as a thick edge from the bottom A node to the bottom P node in Figure 2. As a result, a sequence of a^A_D and a^P_R is determined as a Nash equilibrium[1].

It is theoretically proven that this backward induction process reaches at least one Nash equilibrium in an extensive-form game regardless of players' internal states (e.g., the history of behavior invocations by an agent and its underlying platform) and external states (e.g., network conditions) [4]. Thanks to this stability (i.e., reachability to at least one Nash equilibrium), SymbioticSphere guarantees that agents and platforms deterministically performs symbiotic behaviors in an adaptive and stable manner.

III. Simulation Results

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

Figure 3 shows a simulated cloud data center. It consists of 100 hosts in a grid topology. The grid topology is chosen based on recent findings on efficient topology configurations in data centers [6]. Each agent implements an HTTP service. Users send service requests to agents via the user access point. This paper assumes that a single (virtual) user runs on the access point, and it emulates multiple users to send out service requests. Figure 4 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. The peak workload is 2,500 requests/second. At the beginning of each simulation, one agent and one platform are deployed on a host that is furtherest from the user access point.

This simulation study evaluates the following three variations of SymbioticSphere:

[1]The same backward induction is used to determine a Nash equilibrium in a game for platform-initiated symbiosis. Note that Figure 2 shows a game for agent-initiated symbiosis.

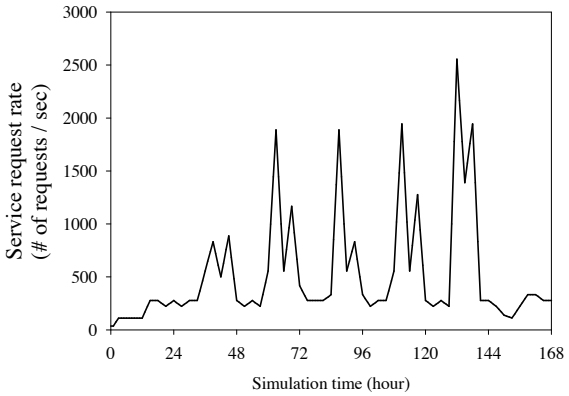

Fig. 4: Request Rate

- **SymbioticSphere-GT (SS-GT):** All agents/platforms periodically play games for agent-initiated and platform-initiated symbiotic behaviors that are described in Section II-D.
- **SymbioticSphere-GTR (SS-GTR):** All agents/platforms periodically play games for agent-initiated and platform-initiated symbiotic behaviors. When an agent and a platform choose a^A_D and a^P_D ("do-nothing") as a symbiotic behavior, they invoke regular behaviors that are described in Sections II-B and II-C.
- **SymbioticSphere-R (SS-R):** All agents and platforms periodically perform regular behaviors. They never invoke symbiotic behaviors.

Fig. 5 shows the average response time that agents in SS-GT, SS-GTR and SS-R yield over time. Fig. 6 shows the average throughput that agents yield over time. At the beginning of a simulation, response time is high (25.45 seconds) because there is only one agent and one platform need to process the entire demand placed on them. As a result, throughput does not reach 100% (51%). However, as agents and platforms replicate themselves and agents migrate toward the user access point, response time drops under four seconds in two hours in SS-GTR, three hours in SS-GT and six hours in SS-R. Throughput improves immediately as well in the beginning of a simulation. Agents yield the throughput of 99% in three hours in SS-GTR, five hours in SS-GT and six hours in SS-R.

After the initial improvement at the beginning of a simulation, response time and throughput constantly remain acceptable in SS-GT and SS-GTR. SS-GT never yields the response time of five seconds or higher. SS-GTR yields the response time of six seconds or higher only once and five seconds or higher six times throughout the entire simulation period (168 hours). SS-GT never yields the throughput of 99% or lower. SS-GTR yields the throughput of 99% or lower 13 times throughout the entire simulation period.

Tables I and II show the minimum, maximum, average, median and standard deviation of hour-by-hour response time and throughput results in Figs. 5 and 6. Table I compares SS-GT and SS-R, while Table II compares SS-GTR and SS-R. SS-GT and SS-GTR outperform SS-R in the minimum,

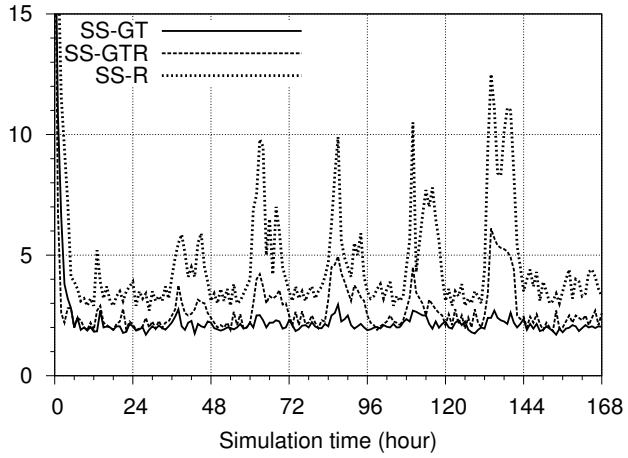

Fig. 5: Response Time (Second)

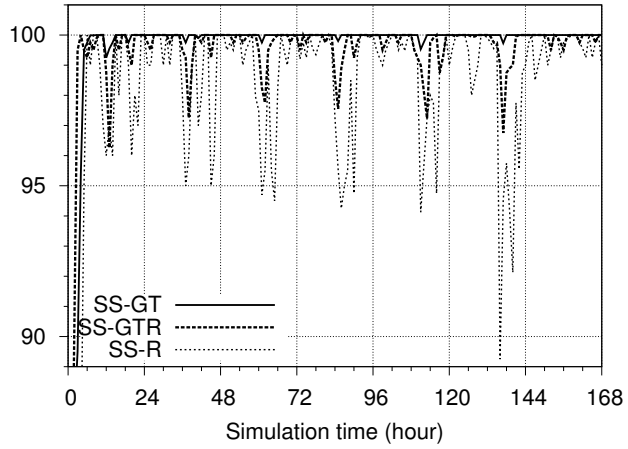

Fig. 6: Throughput (%)

average and median response time. SS-GT and SS-GTR yield 51% and 41% lower response time on average, respectively, than SS-R. Moreover, SS-GT and SS-GTR outperform SS-R in the average and median throughput. SS-GT and SS-GTR yields the median throughput of 100% while SS-R fails to active it.

Superiority of SS-GT and SS-GTR over SS-R comes from symbiotic behaviors. When workload spikes, the number of symbiotic behavior invocations dramatically increases. Particularly, agents and platforms often invoke a symbiotic behavior in which both an agent and a platform replicate themselves. In this symbiotic behavior, agents replicate themselves on their local platforms and platforms replicate themselves on nearby inactive hosts so that they can rapidly increase their availability and effectively process service requests. This allows replicated agents to migrate to replicated platforms to process a higher workload with more resources, thereby improving response time and throughput.

Tables I and II also illustrate the performance stability of

		SS-GT	SS-R	Difference
Response time	Min	1.70	2.70	1.00 (37%)
	Max	20.45	20.45	0.00 (0%)
	Avg	2.30	4.77	2.47 (51%)
	Med	2.05	3.80	1.75 (46%)
	Sd	1.61	3.27	1.66 (51%)
Throughput	Min	51.0	51.0	0.00 (0%)
	Max	100	100	0.00 (0%)
	Avg	99.37	97.87	1.50 (1.5%)
	Med	100	99.50	0.50 (0.5%)
	Sd	4.39	5.40	1.01 (18%)

TABLE I: Comparison between SS-GT and SS-R

		SS-GTR	SS-R	Difference
Response time	Min	1.85	2.70	0.85 (31%)
	Max	20.45	20.45	0.00 (0%)
	Avg	2.81	4.77	1.96 (41%)
	Med	2.47	3.80	1.33 (35%)
	Sd	1.63	3.27	1.64 (50%)
Throughput	Min	51.0	51.0	0.00 (0%)
	Max	100	100	0.00 (0%)
	Avg	99.24	97.87	1.37 (1.3%)
	Med	100	99.50	0.50 (0.5%)
	Sd	4.08	5.40	1.32 (24%)

TABLE II: Comparison between SS-GTR and SS-R

SS-GT, SS-GTR and SS-R as the standard deviation of hour-by-hour response time and throughput results. SS-GT is 51% more stable in response time than SS-R. SS-GTR is 24% more stable in throughput than SS-R.

Fig. 5, Fig. 6, Table I and Table II demonstrate that SS-GT and SS-GTR allow agents and platforms to successfully yield high performance (i.e., low response time and high throughput) by self-adapting their populations and locations against dynamic demand changes. Agents and platforms also effectively leverage symbiotic behaviors to self-stabilize their adaptation decisions, thereby stabilizing their response time and throughput performance.

Fig. 7 shows the resource efficiency in SS-GT, SS-GTR and SS-R. It is computed as $\frac{S}{R}$ where S denotes the total number of service requests that agents process in the entire data center and R denotes the total amount 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 a less amount of resources. In both SS-GTR and SS-R, resource efficiency follows the changes in service request rate (Fig. 4). Given the throughput performance in Fig. 6, this means that agents and platforms self-adapt their availability (i.e., the number of replicas) and in turn resource utilization as service request rate changes.

SS-GTR and SS-R yield qualitatively similar resource efficiency; however, SS-R's resource efficiency is often a little higher than SS-GTR's. This is because (1) SS-R's throughput is lower than SS-GTR's (Fig. 6) and (2) SS-GTR encourages agents and platforms to cooperate with symbiotic behaviors to survive longer, thereby maintaining their higher availability than in SS-R. Higher availability means higher resource utilization; i.e., higher R.

SS-GT's resource efficiency consistently remains low be-

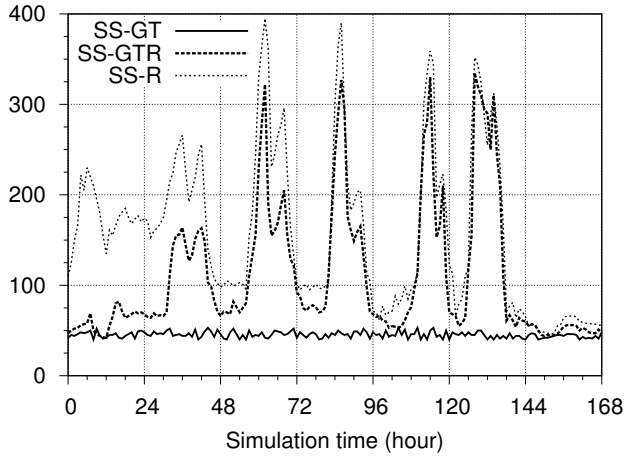

Fig. 7: Resource Efficiency

cause agents and platforms never die while they can replicate themselves. Note that symbiotic behaviors do not consider the death of agents and platforms. Agents and platforms never invoke regular behaviors including the death behavior in SS-GT. This means that, in SS-GT, the availability of agents and platforms can increase but never decrease. This property allows SS-GT to outperform SS-GTR in response time and throughput (Figs. 5 and 7).

IV. Related Work

This paper extends the authors' prior work [7], [8]. This paper investigates symbiotic behaviors as well as regular behaviors, while only regular behaviors are studied in [7]. Symbiotic behaviors are studied in [8]. However, in [8], 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. Stabilizability in adaptive behavior invocation is not studied in [7], [8].

Many GAs have been used for adaptation of cloud/grid applications (e.g., [9], [10]). They seek the optimal adaptation solutions; it is out of their scope to seek equilibrium solutions. They do not consider stability in adaptation. In SymbioticSphere, agents and platforms seek equilibria in their adaptation as symbiotic behaviors.

Game theoretic algorithms have been used in several aspects of cloud/grid applications; for example, task allocation [11], application placement [12]–[14] and data replication [15]. [11] maintains stability in seeking equilibria; however, it assumes static networks whose conditions (e.g., network traffic) never change over time. [12], [13] 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. [14], [15] 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.

V. Concluding Remarks

This paper describes a game theoretic adaptation mechanism in SymbioticSphere and evaluates its impacts on the self-adaptation 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) with limited performance fluctuations.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," University of California, Berkeley, EECS Dept., Tech. Rep., 2009.

[2] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: Elastic resource scaling for multi-tenant cloud systems," in *Proc. ACM Symposium on Cloud Computing*, 2011.

[3] A. Gulati, G. Shanmuganathan, I. Ahmad, and A. Holler, "Cloud scale resource management: Challenges and techniques," in *Proc. USENIX Workshop on Hot Topics in Cloud Computing*, 2011.

[4] R. Cressman, *Evolutionary Dynamics and Extensive Form Games*. MIT Press, 2003.

[5] N. Srinivas and K. Deb, "Multiobjective function optimization using nondominated sorting genetic algorithms," *Evol. Computat.*, vol. 2, no. 3, 1995.

[6] C. Guo, H. Wu, K. Tan, L. Shiy, Y. Zhang, and S. Lu, "DCell: A scalable and fault-tolerant network structure for data centers," in *Proc. ACM Int'l SIGOMM Conference*, 2008.

[7] P. Champrasert and J. Suzuki, "Symbioticsphere: A biologically-inspired autonomic architecture for self-adaptive and self-healing server farms," in *Proc. IEEE Int'l Workshop on Autonomic Communications and Computing*, 2006.

[8] ——, "Building self-configuring data centers with cross layer coevolution," *Journal of Software*, vol. 2, no. 5, 2007.

[9] H. Wada, J. Suzuki, Y. Yamano, and K. Oba, "Evolutionary deployment optimization for service oriented clouds," *Software: Practice and Experience*, vol. 41, no. 5, 2011.

[10] Q. Tang, S. Gupta, and G. Varsamopoulos, "Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: a cyber-physical approach," *IEEE T. Parall. Distr. Syst.*, vol. 19, no. 11, 2008.

[11] R. Subrata, A. Zomaya, and B. Landfeldt, "Game-theoretic Approach for Load Balancing in Computational Grids," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 1, 2008.

[12] G. Wei, A. V. Vasilakos, Y. Zheng, and N. Xiong, "A game-theoretic method of fair resource allocation for cloud computing services," *J. Supercomput.*, vol. 54, no. 2, 2010.

[13] 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," *Computer Communications*, vol. 30, no. 3, 2007.

[14] C. Lee, J. Suzuki, A. V. Vasilakos, Y. Yamano, and K. Oba, "An evolutionary game theoretic approach to adaptive and stable application deployment in clouds," in *Proc. of ACM Workshop on Bio-Inspired Algorithms for Distr. Syst.*, 2010.

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