# Online Self-Reconfiguration with Performance Guarantee for Energy-Efficient Large-Scale Cloud Computing Data Centers

**Conference Paper** · August 2010

DOI: 10.1109/SCC.2010.69 · Source: IEEE Xplore

CITATIONS
111

READS
333

**6 authors**, including:

Haibo Mi
National University of Defense Technology
**17** PUBLICATIONS **234** CITATIONS

SEE PROFILE

Dianxi Shi
Wuhan University
**31** PUBLICATIONS **208** CITATIONS

SEE PROFILE

# Online Self-reconfiguration with Performance Guarantee for Energy-efficient Large-scale Cloud Computing Data Centers

Haibo Mi[1], Huaimin Wang[1], Gang Yin[1], Yangfan Zhou[2], Dianxi Shi[1], Lin Yuan[1,3]

*1.School of Computer Science, National University of Defense Technology, Changsha, China*
*2.Dept. of Computer Science & Engineering, The Chinese Univ. of Hong Kong, Shatin, Hong Kong*
*3. School of Electronic Technology, Information Engineering University, Zhengzhou, China*

## Abstract

*In a typical large-scale data center, a set of applications are hosted over virtual machines (VMs) running on a large number of physical machines (PMs). Such a virtualization technique can be used for conserving power consumption by minimizing the number of PMs that should be turned on according to the application requirements to resource. However, the resource demands for VMs is dynamic in nature since the number of user requests the applications should handle is rapidly changing in practice. It is a great challenge to online reconfigure the VMs (i.e., optimize the number and the locations for the VMs) according to the dynamic resource demands. Especially for the emerging applications of large-scale data centers for cloud computing systems, existing approaches either fails to find a best configuration of VMs or cannot produce a result in an acceptable time. In this paper, we propose an online self-reconfiguration approach for reallocating VMs in large-scale data centers. It first accurately predicts the future workloads of the applications with Brown's quadratic exponential smoothing. Based on such a prediction, it adopts a genetic algorithm to efficiently find the optimal reconfiguration policy. The resource utilization of large-scale cloud computing data centers can thus be improved and their energy consumption can be greatly conserved. We conduct extensive experiments and the results verify that our approach can effectively switch off more unnecessary running PMs comparing with current approaches without a performance degradation of the whole system.*

## 1. Introduction

Energy consumption in large-scale data centers is one of major concerns for achieving green computing and cost-down. But according to a recent survey [1], it is still the second-highest operating cost for data centers. The root cause of such huge energy consumption is the low average utilization of resources [2]. Maximizing the average resources utilization is hence a natural means to achieve energy-efficient cloud computing data centers. Typical data centers generally consolidate multiple applications to share the same physical resources through virtualization technology [6]. However, traditional ways of static VMs consolidation according to application peak demands suffer bad resource utilization. A more efficient solution is to dynamically decide the running locations of different VMs in response to their time-varying resource requirements caused by the dynamics of user requests. However, such a VM self-reconfiguration problem is computationally hard to solve, especially for large-scale data centers. This poses a great obstacle for online optimizing the resource utilization to achieve energy efficiency.

Recently, there have been many studies trying to realize self-reconfiguration of data centers so as to maximize resources utilization and conserve energy consumption [2-10, 18, 24]. However, most approaches can only be applied in small scale cluster because of the poor scalability of the algorithms. Moreover, they largely consider the case that there is only one VM for each application [3, 5-7]. This consideration is not adequate for modern practical data centers for cloud computing systems, where the same application can be hosted in multiple VMs which are deployed in different PMs to achieve elasticity [24].

In this paper, we propose a genetic algorithm based approach, namely GABA, to adaptively self-reconfigure the VMs in virtualized large-scale data centers consisting of heterogeneous nodes. GABA can efficiently decide the optimal number of VMs online for each application and their physical locations according to time-varying requirements and the dynamic environmental conditions. It has two merits: First, it enables data centers self-reconfigure without explicit specifications. Second, it can efficiently online search the optimal solutions through the vast and complex possible reconfiguration spaces.

We verify our approach by experiments on a simulated environment of a large-scale data center consisting of 300 PMs. Comparing with the most recent approach proposed in [10], the experimental results show

that our approach can effectively switch off much more unnecessary running and increase the average resource utilization. Energy consumption is thus greatly reduced.

The rest of this paper is organized as follows. Section 2 presents the related work. In Section 3, we introduce the architecture of self-reconfiguration management prototype where our approach is implemented. Section 4 discusses the design details of GABA. In Section 5, we present our experimental studies. Finally, Section 6 concludes the paper and points out some possible future directions.

## 2. Related Work

Recently, extensive research efforts have been put in realizing self-reconfiguration of data centers to maximize resource utilization for conserving energy consumption [2-10, 18]. Walsh et al. [3] proposed a utility function-based approach to dynamically self-manage data centers with heterogeneous nodes. Two-level utility functions are designed to allocate resources. A service level utility function is used to compute application resource demand while a resource level one is responsible for allocating resources among applications. This approach was applied to design a prototype of power and performance management system based on multi-agent technology in [4]. Based on decompositional reinforcement learning [8], Tesauro et al. proposed to achieve online resource allocation without developing explicit system models [5]. They regard the allocation problem as a composite MDP [16], which treats the cross product of the local application state spaces as the state space, and the set of joint allocations as the action space. However, these approaches do not allow independent applications to be hosted over the same PMs, which is not suitable for virtualized environments typically adopted by cloud computing data centers.

Bobroff et al. [6] proposed a dynamic server migration and consolidation approach to reduce the amount of unnecessary running PMs. The resource demands of VMs are estimated with the forecasting method proposed in [17] and sorted in descending order. Moreover, a heuristic based on the first-fit bin packing approximation is applied to allocate resources in VMs on the constraints of exceeding capacity of PMs to some extent. However, this approach just considers the scenario of one VM for each application, which is not suitable to the scenarios where VMs need multiple copies to deal with user requests.

Based on the limited lookahead control technology [9], Kusic et al. [7] proposed a dynamic resource provisioning framework for virtualized server environment and developed a two-level control hierarchy which separately decided the resource share of VMs in the same PM and the number of PMs provisioning to the same type of VMs. Although the approach improves the resource utilization through switching off unnecessary PMs in the cluster, it suffers poor scalability because of the exponential relationship between the computational complexity and the system size.

Karve et al. [18] proposed a dynamic application instance placement approach for large-scale clustered applications in response to time-varying resource demands while allowing applications to share PMs. They formulated the problem as one with multiple optimization objectives and solved it by a simple heuristic algorithm. Tang et al. [10] further improved this approach in terms of computational complexity. These algorithms, though can be applied in virtualized environment, they try to maximize the satisfied application demands, but do not consider the case of shutting down idle machines in order to reduce energy consumption. Our approach, in contrast, considers switching off unnecessary PMs, and consequently outperforms these existing approaches in terms of power saving while keeping high efficiency.

## 3. Self-reconfiguration for Cloud Computing Data Centers in a Nutshell.

In cloud computing data centers, a set of applications are generally hosted over VMs running on a large number of PMs. We consider that there are $N$ applications $\{A_i\}_{i=1}^{N}$, corresponding to $N$ types of VMs $\{V_i\}_{i=1}^{N}$. Assume that $A_i$ is running over $V_i$. $V_i$ can be loaded in multiple PMs based on its resource requirements. Let $n_i$ denote the number of copies of $V_i$ loaded in multiple PMs, i.e., the number of PMs that should load $V_i$ to host $A_i$. A self-reconfiguration management framework is one that is employed to dynamically decide $n_i$ and which PM a copy of $V_i$ should be loaded for hosting application $A_i$ according to the time-varying requirements of the application.

Figure 1 shows such a self-reconfiguration management framework considered in this paper. We consider a large-scale data center consisting of many heterogeneous PMs on which multiple types of VMs may be deployed. As shown in the figure, the front-end of the framework contains three key components: Dispatcher, VM Managers (VMMs), and Reconfiguration Policy Generator (RPG). Dispatcher is for delivering a user request of an application to its corresponding VM manager (VMM). Once there is a new kind of applica-

tion deployed on a VM in the cluster, the front-end generates a new VMM for this kind of VM.

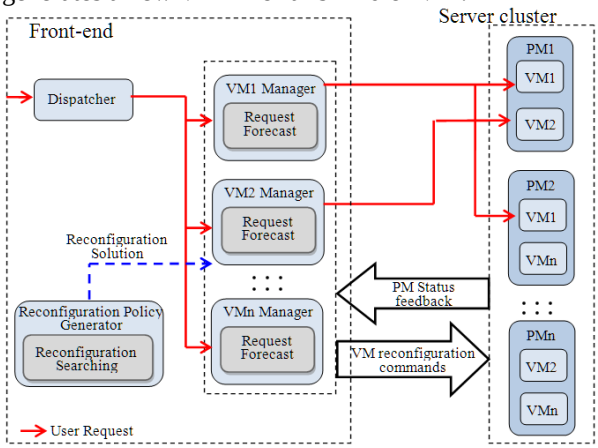


**Figure 1. Self-reconfiguration architecture**

Since it takes seconds to adopt reconfiguration, in order to make the reconfiguration solution catch up with demands, VMMs use the Request Forecast module to estimate the demands of the future workload (i.e. the number of user requests). Then VMMs send the predictive value to RPG. According to the predictions, RPG uses Reconfiguration Searching module to find the optimal solution through possible reconfiguration spaces. The reconfiguration solution is then sent to VMMs. This serves as a guide line for VMMs to migrate/create the VMs among the PMs and turn off/on the PMs for best performance and energy efficiency. Lastly, besides managing the VMs, VMMs are also responsible for further dispatching a user request to a particular VM.

We can see that the Request forecast module in VMMs and the Reconfiguration Searching module in RPG are core to this reconfiguration framework. In this paper, we design a Brown's quadratic exponential smoothing method for estimating the future workload of the VMs. We also specifically tailor a genetic algorithm to implement the Reconfiguration Searching module.

# 4. The GABA Approach

In this section, we will present the design details of GABA approach.

## 4.1 Problem Formulation

The problem of finding the optimal reconfiguration solution to increase resources utilization of PMs as well as to conserve power consumption can be dealt as multiple optimization objectives. In this paper we just consider resource as CPU. For every resource allocation the formal statement of this problem is expressed in formula (1).

$$\begin{aligned} &Min \quad F_{power} \\ &Max \quad \overline{U_{CPU}} \\ &s.t: \\ &\quad \sum_{i=1}^{N} r_{ij} = load_j(t+\tau) \quad \forall j \in M \\ &\quad \sum_{j=1}^{M} \frac{r_{ij}}{req_{ij}} \le 1 \quad \forall i \in N \end{aligned} \tag{1}$$

Where $r_{ij}$ is the request for VM $j$ deployed on PM $i$ and $\sum_{i=1}^{N} r_{ij}$ represents the sum of requests in the cluster, which should equal to the predictive demand values $load_j(t+\tau)$ . $\frac{r_{ij}}{req_{ij}}$ represents CPU demand of VM $j$ for PM $i$ and $\sum_{j=1}^{M} \frac{r_{ij}}{req_{ij}}$ denotes the sum of CPU demand of all VMs deployed on PM $i$ ,which should observe the capacity limits of PM $i$ . The design details of $F_{power}$ , $\overline{req_{ij}}$ and $\overline{U_{CPU}}$ will be discussed below.

## 4.2 Request Forecasting

Because it takes seconds for VM to start or stop, it is necessary to predict resource requirement for VMs to make reconfiguration catch up with demands. The forecasting approach used here is based on Brown's quadratic exponential smoothing method, a predictive technique developed in [11]. The forecasting formula can be defined as:

$$\begin{cases} load_j(t+\tau) = a_t + b_t, \tau = 1,2,3... \\ a_t = 2L_t^{(1)} - L_t^{(2)} \\ b_t = \dfrac{\alpha}{1-\alpha}(L_t^{(1)} - L_t^{(2)}) \\ L_t^{(1)} = \alpha * load_j(t) + (1-\alpha) * L_{t-1}^{(1)} \\ L_t^{(2)} = \alpha * L_t^{(1)} + (1-\alpha) * L_{t-1}^{(2)} \end{cases} \tag{2}$$

Where $load_j(t)$ represents the request numbers asking for VM $j$ at time interval $t$ . $load_j(t+\tau)$ represents the predictive value at time interval $t+\tau$ . $\tau$ is monitor interval. $L_t^{(1)}$ and $L_t^{(2)}$ are denoted as simple and quadratic exponential smoothing value respectively. $\alpha$ is smoothing parameter.

### 4.3 GA-based Reconfiguration Approach

#### 4.3.1 Chromosome Coding

Table 1. PMs response ability to three kinds of VMs

| PM | Req/s for VM1 | length of coding | Req/s for VM2 | length of coding | Req/s for VM3 | length of coding |
|---|---|---|---|---|---|---|
| 1 | 100 | 7 | 200 | 8 | 300 | 9 |
| 2 | 200 | 8 | 300 | 9 | 400 | 9 |

Since the multiple optimization objectives problem solved by Reconfiguration Searching Model is NP-hard, if we try to online adjust configurations to changing environmental conditions, the optimal policy must be searched through reconfiguration spaces as soon as possible in terms of new demands. The trait of high efficiency of genetic algorithm to search optimal results makes us apply it to find best solutions.

Generally speaking, the application deployed on VM $j$ may contain many different types of request, such as ordering, shopping, browsing, etc. Different types of requests may demand different computational resources, and it will become complex if VMs are scheduled according to request-level grain. Therefore, application-level schedule is promoted. Supposed one VM can only deploy one kind of application. For VM $j$, the average request numbers that PM $i$ can respond to for every second can be defined as:

$$\overline{req_{ij}} = \sum_{q \in Q_j} \alpha_q * req_{iq} \tag{3}$$

Where $Q_j = \{q_1, q_2, ..., q_n\}$ is the set of all kinds request types for VM $j$. $req_{iq}$ represents the maximum numbers that PM $i$ can respond to the type $q$ request and $\alpha_q$ is the fraction of type $q$ request in all requests.

The specification of Chromosome coding is as follows. Every chromosome represents a candidate reconfiguration and composed of genetic strings. The number of genetic strings is determined by the kind of VMs. Each genetic string denoting one kind of VMs contains of some sub-strings. For VM $j$, every its sub-string corresponds to one PM. The length of sub-string is defined by the length of binary code of the average request number that the PM can respond to VM $j$, while the length of genetic string is determined by the sum of length of sub-strings. For example, there are two kinds of PMs, whose response ability to VM $j$ are 100 requests/s and 200 requests/s respectively. Therefore, the lengths of the sub-strings corresponding to them are 7 bits and 8 bits, and the length of the genetic string of VM $j$ is 15 bits. Figure 2 shows a chromosome contains three kinds of VMs under the assumption shown in Table 1.

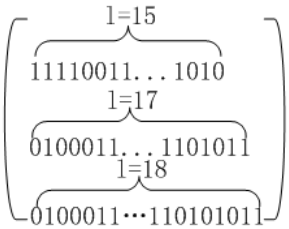


**Figure 2. An example of Chromosome coding**

#### 4.3.2 Fitness Functions

In this paper fitness function is composed of two parts: power consumption function and punish function. Every resource provision, we pursue the minimum of the cluster power consumption. Power consumption function for PM $i$ is defined as:

$$F_{Power\_i} = \begin{cases} \tau * \left( \sum_{j=1}^{M} \frac{r_{ij}}{req_{ij}} P_{Bi} + (1 - \sum_{j=1}^{M} \frac{r_{ij}}{req_{ij}}) P_{Ii} \right) & if\ \exists r_{ij} \neq 0 \\ \tau * P_i^{off} & else \end{cases} \tag{4}$$

Where $P_{Bi}$, $P_{Ii}$ and $P_i^{off}$ are denoted as PM $i$'s busy, idle and asleep power respectively. If PM $i$ has no VM deployed on it, we set it to sleep, in this situation power is computed as $\tau * P_i^{off}$. $M$ is the number of different types of VMs. Therefore, the power of the cluster can be computed as $F_{power} = \sum_{i=1}^{N} F_{Power\_i}$, Where $N$ is the number of PMs.

To efficiently accelerate the process of searching to the optimal, we apply punish function to conduct searching into possible solution spaces. Let $\alpha_j$ be punish parameter of VM $j$. Punish function is defined as:

$$F_{punish} = \sum_{j=1}^{M} F_{punish\_j} = \sum_{j=1}^{M} \alpha_j * \frac{\tau * \sum_{i=1}^{N} \frac{\left| load_j(t+\tau) - \sum_{i=1}^{N} r_{ij} \right|}{req_{ij}} * P_{Bi}}{N} \tag{5}$$

The average CPU utilization of running PMs is defined as:

$$\overline{U_{CPU}} = \frac{\sum_{i \in N} \sum_{j=1}^{M} \frac{r_{ij}}{req_{ij}}}{\left| \sum_{i \in N} i \right|} \tag{6}$$

where $\left| \sum_{i \in N} i \right|$ represents sum of running PMs.

**Algorithm 1 GA-based reconfiguration searching**

```
1:  Initialize population
2:  for each generation in generations
3:    for each chromos in population
4:      for (i=1;1<=chromos.sizes;i++)
5:        str=String.valueof(chromos.get(i));
6:        for (j=1;j<servers.length;j++)
7:          server_load[j][i]=str.substring(
                        serverType.length)
8:        end for
9:      end for
10:     computing fitness for each chromos
11:   end for
12:   selection;
13:   crossover;
14:   mutation;
15: end for
```

Let $U_{CPU\_up}$ and $U_{CPU\_down}$ be the lowest and highest threshold of average CPU utilization. In order to keep high CPU utilization instead of exceeding PM's resource capacity limits. Fitness function is defined as:

$$F_{fitness} = \begin{cases} C_{\max} - F_{power} - F_{punish} & if\ \overline{U_{CPU}} \in [U_{CPU\_down}, U_{CPU\_up}] \\ C_{mix} & others \end{cases} \quad (7)$$

Where $C_{mix}$ and $C_{\max}$ are constants. Algorithm 1 presents the pseudo code of GA-based reconfiguration searching. server_load[j][i] represents the requests for VM $i$ allocated to PM $j$ and the values of server_load denote final reconfiguration result when the algorithm come to convergence.

# 5. Experiments

## 5.1 Experimental Setup

Experiments are simulated in the environment of cluster with 300 heterogeneous PMs, which include 100 Dells with Intel(R) Core(TM)2 Duo 2.83GHz, 100 Dells with Intel(R) Core(TM)2 Duo 2.33GHz and 100 lenovos with AMD Athlon(tm) 64 X2 3600+ 1.9GHz. VM adopts VMWare Workstation. Operating system in VM adopts Red Hat 2.6.24.3. Application Server adopts Tomcat6.0. Power consumption of each type of PM is listed in Table 2.

Table 3 lists the default values of key parameters which must be considered in GABA approach. Different values of these parameters have different influence on the effect and efficiency of the approach. How to select the most suitable values for these parameters is beyond the scope of this paper. This paper just discusses generation times for each reconfiguration below.

**Table 2. PM power consumption**

| PM | $P_{busy}$(W) | $P_{idle}$(W) | $P_{off}$(W) |
|---|---|---|---|
| Dell(2.83GHz) | 268 | 155 | 10 |
| Dell(2.33GHz) | 225 | 112 | 10 |
| lenovo(1.9GHz) | 189 | 101 | 10 |

**Table 3. Values of key parameters**

| Parameter | value |
|---|---|
| Population size | 50 |
| Selection method | Roulette wheele[23] |
| Crossover rate | 0.9 |
| Mutation rate | 0.1 |

## 5.2 Workload Generation

### 5.2.1 Workload Types

In experiments, three types of VM are simulated. Each type of applications deployed in the VM is referred to TPC-W benchmark [12], those are Browsing Mix (BM), Shopping Mix (SM) and Ordering Mix (OM) respectively. We use the tool of Httperf [13] to do stress test for three kinds of PMs. Table 4 lists the application-level response capability of different kinds of PMs for a single type of VM.

**Table 4. PM application-level response capability**

| PM | Req/s(BM) | Req/s(SM) | Req/s(OM) |
|---|---|---|---|
| Dell(2.83GHz) | 235 | 197 | 130 |
| Dell(2.33GHz) | 190 | 163 | 91 |
| lenovo(1.9GHz) | 132 | 89 | 57 |

### 5.2.2 Workload Traces

Totally we need three kinds of time-varying resource demand for three types of VMs. Two of them are based on workload traces of request to France world Cup 1998 web site on June 22 [14] and ClarkNet web server[15] on September 29 respectively. Both of the data start on 12:00AM and end on 24:00PM. Note in order to make sufficient workload, both of traces are enlarged by 10 times. The third is generated according to Poisson distribution. The data are sampled every 10 minutes. Figure 5 shows three kinds of workload traces.

### 5.3 Experimental results

In this section, we mainly evaluate the efficiency and effectiveness of the GA-based self-reconfiguration

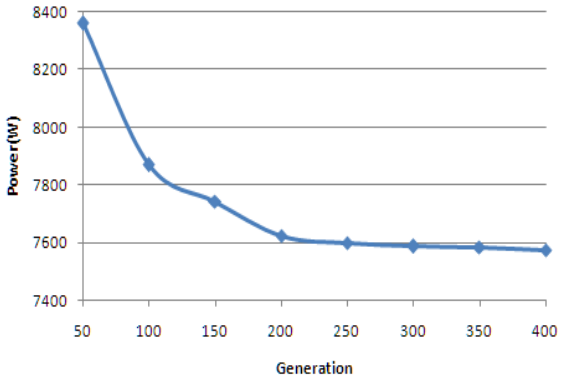


**Figure 3. Convergence of GA-based self-reconfiguration approach**

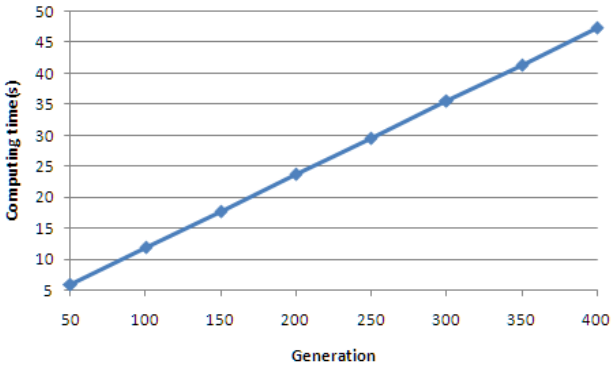


**Figure 4. Computing time for different generations**

approach. Every result is the average of 10 trials of the experiment.

### 5.3.1 Convergence of Algorithm

It is very important for a GA-based approach to steadily converge in time. As Figure 3 and Figure 4 show, the GABA approach comes to convergence by 250 generations and it takes about 28s of computing time, whereas in [7], it requires about 2mins of execution time to find a suitable configuration. Moreover, from Algorithm 1, we can see the complexity of the GABA approach is influenced by the number of PMs, the kinds of VMs, initial population size, and generations. Once the other three parameters are assigned to fixed values, the computing time of the GABA approach is almost linearly increasing with the increase of generations, as is shown in Figure 4, which confirms that our approach has much high scalability.

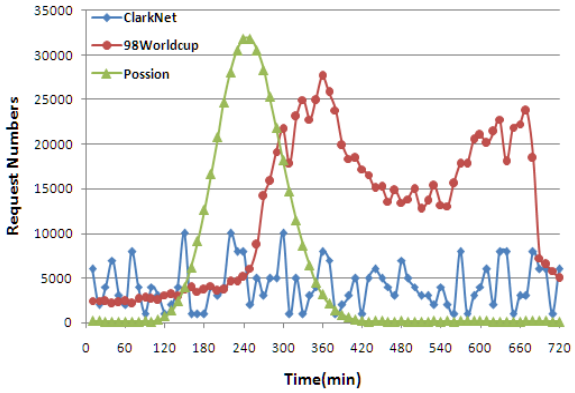


**Figure 5. Distributions of three kinds of requests**

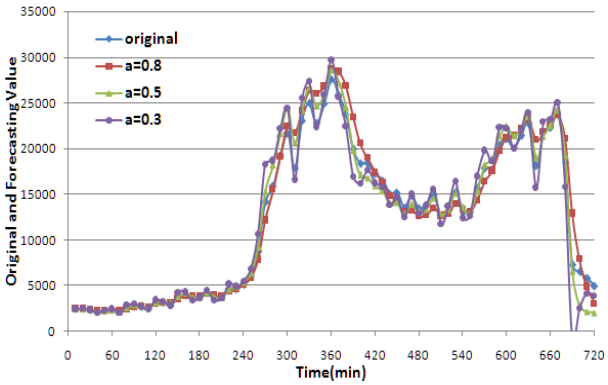


**Figure 6. Accuracy of the predictive results**

### 5.3.2 Forecasting Result

Taking the data of 98Worldcup for example, the contrast between original and predictive request is illustrated in Figure 6. The smoothing parameter $\alpha$ is assigned to real value between 0.1 and 1. As shown in the figure, the Brown's quadratic exponential smoothing method produces good estimates with a small amount of error. Figure 6 plots the original request and predicting request when $\alpha$ equals 0.3, 0.5, and 0.8. Through analyzing the mean relative error (MRE), we can know the MRE values are 0.078, 0.064, and 0.1 respectively. Therefore, when $\alpha = 0.5$, the predictive result is best.

### 5.3.3 Performance of Reconfiguration

In this section we study the effectiveness of the GABA approach for online self-reconfiguration. We compare the approach with the one proposed in [10], TSSP07 for short. Figure 7, 8, 9 show the contrast results of running nodes, CPU utilization and power

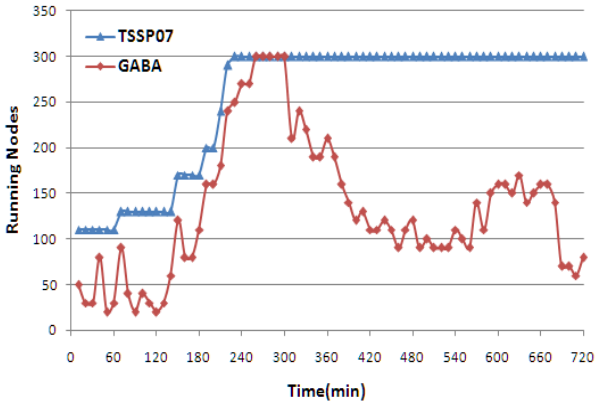


**Figure 7. Comparison of the number of running nodes**

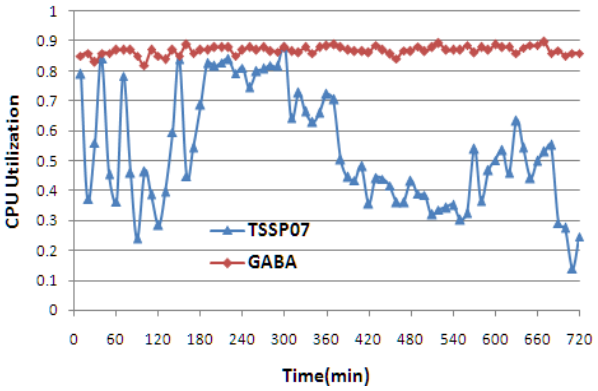


**Figure 8. CPU utilization comparison**

consumption in 12 hours respectively. As Figure 7 shows, GABA performs better than TSSP07 as it starts many unnecessary PMs in the whole periods. Especially in order to reduce disturbance to the cluster, TSSP07 don't adjust the cluster when requests can be satisfied by current configuration. Therefore, when the demand comes to peak at 300 minute, TSSP07 starts all PMs. However, the approach still keeps all PMs running even after the request decreases. Totally, the average number of running nodes in 12 hours is 255 with TSSP07 while the average number is 133 with GABA, which is just approximately 1/2 as many as TSSP07.

Because of those unnecessary running PMs, the average CPU utilization in this periods is 53.4% when adopting TSSP07. There is approximately 35% utilization improvement from GABA than TSSP07, as is shown in Figure 8. Figure 9 plots the power consumption of two approaches, TSSP07 consumes totally of 536.2 kilowatt, while GABA consumes nearly 409.3 kilowatt, which conserves the consumption by up to 25%.

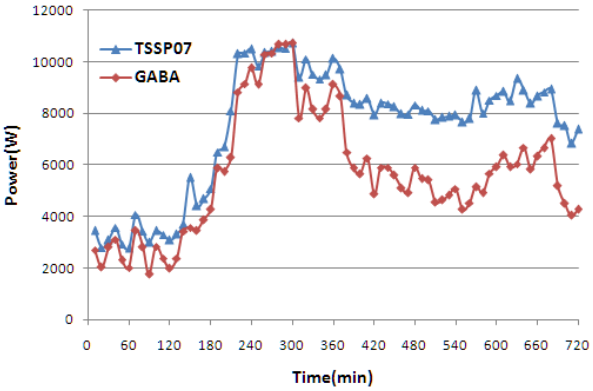


**Figure 9. Power consumption comparison**

## 6. Conclusion

Energy efficiency is becoming a more and more urgent concern for large-scale data centers to achieve green computing and cost-down. Increasing the average resource utilization is critical technique for achieving energy-efficiency. In this paper, we propose a standard genetic algorithm based online self-reconfiguration approach, namely GABA, for large-scale data centers. GABA can effectively online adjust the number of VMs for different applications and their physical locations according to time-varying resource requirement and environmental conditions. By optimizing the number and locations of the VMs, GABA can minimize the number of PMs that should be turned on. This can greatly improve PM utilization and thus conserve energy. We demonstrate by extensive experiments that comparing with existing approaches GABA can effectively switch off more unnecessary running PMs without a performance degrading of the whole system.

GABA adopts genetic algorithm (GA) for searching the optimal reconfiguration policy. GA is widely adopted in a lot of practical fields, e.g., data mirroring [20], reconfiguration of network [21], network coding [22], etc. It is a popular fast and effective search technique for finding exact or approximate solutions to optimization and searching problems [19], which is best suitable for solving our reconfiguration problem in large-scale cloud computing data centers. Also, it is worth noting that GA is not the sole option for searching the optimal solutions. Other general algorithms (e.g., ant colony optimization and simulate anneal) may also be applied. Their pros and cons will be studied in our future work.

Finally, this work assumes different applications deployed on VMs are independent of each other. This is generally valid for current applications hosted in cloud computing data centers. But there are still cases that the applications may be relevant, making the searching

space larger. In our future work, we are particularly interested in tailoring GABA to adapt to such cases.

## References


[1] J.G. Koomey. Estimating total power consumption by servers in the U.S. and the world. http://enterprise.amd.com/Downloads/svrpwrusecompletefinal.pdf, 2007.

[2] P. Padala, K.G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, K. Salem. Adaptive control of virtualized resources in utility computing environments. *ACM SIGOPS Operating Systems Review*, 41(3): 289-302, 2007.

[3] W.E. Walsh, G. Tesauro, J.O. Kephart, R. Das. Utility functions in autonomic systems. In *Proc. of the 4th International Conference on Autonomic Computing*, pages 70-77, 2007.

[4] R. Das, J.O. Kephart, C. Lefurgy, G. Tesauro, D. W. Levine, H. Chan. Autonomic multi-agent management of power and performance in data centers. In *Proc. of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 107-114, 2008.

[5] G. Tesauro. Online resource allocation using decompositional reinforcement learning. In *Proc. of 20th national Conference on Artificial Intelligence*, pages 886-891, 2005.

[6] N. Bobroff, A. Kochut, K. Beaty. Dynamic placement of virtual machines for managing SLA violations. In *Proc. of 10th IFIP/IEEE International Symposium on Integrated Network Management*, pages 119-128, 2007.

[7] D. Kusic, J.O. Kephart, J.E. Hanson, N. Kandasamy, G. Jiang. Power and performance management of virtualized computing environments via lookahead control. *Journal of Cluster Computing*, 12(1): 1-15, 2009.

[8] S. Russel, A.L. Zimdars. Q-Decomposition for Reinforcement Learning Agents. In *Proc. of the Twentieth International Conference on Machine Learning*, pages 656-661, 2003.

[9] S. Abdelwahed, N. Kandasamy, S. Neema. Online control for self-management in computing systems. In *Proc. of Real-Time and Embedded Technology and Applications Symposium*, pages 368-375, 2004.

[10] C. Tang, M. Steinder, M. Spreitzer, G. Pacifici. A scalable application placement controller for enterprise data centers. In *Proc. of the 16th International Conference on World Wide Web*, pages 331-340, 2007.

[11] R.G. Brown, R.F. Meyer. The fundamental theorem of exponential smoothing. *Journal of Operations Research*, (9): 673-685, 1961.

[12] D.A. Menasc. TPC-W: A benchmark for e-commerce. *IEEE Internet Computing*, 6(3): 83-87, 2002.

[13] D. Mosberger, T. Jin. Httperf-a tool for measuring web server performance. *ACM SIGMETRICS Performance Evaluation Review*, 26(3): 31-37, 1998.

[14] M. Arlitt, T. Jin. Workload characterization of the 1998 world cup web site. In *Technical Report HPL-99-35R1Hewlett-Packard Labs*, 1999.

[15] M. Arlitt, C. Williamson. Web server workload characterization: The search for invariants. In *Proc. of ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems*, pages 23-26, 1996.

[16] N. Meuleau, M. Hauskrecht, K.E. Kim, et al. Solving very large weakly coupled markov decision processes. In *Proc. of 13th National Conference on Artificial Intelligence*, pages 165-172. 1998.

[17] G. Jenkins, G. Reinsel, G. Box. *Time Series Analysis: Forecasting and Control*. Prentice Hall, 1994.

[18] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, A. Tantawi. Dynamic placement for clustered web applications. In *Proc. of the 15th International Conference on World Wide Web*, pages 593-604, 2006.

[19] N. Chaiyaratana, A.M.S. Zalzala. Recent developments in evolutionary and genetic algorithms: theory and applications. In *Proc. of the second International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pages 270-277, 1997.

[20] A.J. Ramirez, D.B. Knoester, B.H. Cheng, P.K. McKinley. Applying genetic algorithms to decision making in autonomic computing systems. In *Proc. of the 6th International Conference on Autonomic computing*, pages 97-106, 2009.

[21] D. Montana, T. Hussain. Adaptive reconfiguration of data networks using genetic algorithms. *Journal of Applied Soft Computing*, 4(4): 433-444, 2004.

[22] L. Deng, J. Zhao, X. Wang. Genetic algorithm solution of network coding optimization. *Journal of Software*, 20(8): 2269-2279, 2009.

[23] D.B. Fogel. An introduction to simulated evolutionary optimization. *IEEE transactions on Neural Networks*, 5(1):3-14, 1994.

[24] C. Weng, M. Li, Z. Wang, X. Lu. Automatic Performance Tuning for the Virtualized Cluster System. In Proc. of the 29th IEEE International Conference on Distributed Computing Systems-Volume 00, pages 183-190, 2009