# Toward A Genetic Algorithm Based Flexible Approach for the Management of Virtualized Application Environments in Cloud Platforms

Omar Abdul-Rahman
Graduate School of Information Science & Technology
Hokkaido University
Sapporo, Japan
omar@ist.hokudai.ac.jp

Masaharu Munetomo and Kiyoshi Akama
Information Initiative Center
Hokkaido University
Sapporo, Japan
munetomo@iic.hokudai.ac.jp, akama@iic.hokudai.ac.jp

***Abstract*— Resource management in cloud platforms becomes an increasingly complex and daunting task surrounded by various challenges of stringent QoS requirements, service availability guaranteeing and escalating overhead of the infrastructure that resulted from operation costs and ecological impact. On the other hand, virtualization adds a greater flexibility to the resource managers in addressing such challenges. However, at the same time, it imposes a further challenge of added management complexity. Recently, we have proposed a resource management model for cloud platforms, which utilizes a new resource mapping formulation and relays on a hybrid virtualization framework in an attempt to realize a resource manager that intelligently adapts the available cloud resources to satisfy the conflicting objectives of the running applications and underlying infrastructures' requirements. Moreover, we have proposed state of the art Binary-Real coded Genetic Algorithm (BRGA), which has been applied successfully to a wide spectrum of global and constrained optimization problems from the known benchmark suites. In this paper, we aim to proceed by proposing a mathematical model and a modified version of BRGA to validate our model. In addition, we aim to evaluate the feasibility, effectiveness and scalability of our approach through simulation experiments.***

***Keywort; Service Oriented Architecture (SOA); Autonomic Computing; Decentralization; parallelization; Genetic Algorithm***

## I. INTRODUCTION

Cloud computing is a promising emerging paradigm that enables enterprises to host multiple of independent applications on a shared resource pool with the ability to lease computational power in the form of virtual machines on the per-demand basis. [1]

On the other hand, virtualization is an enabling technology that adds a great flexibility to the resource managers in addressing such challenges. By employing virtualization, the dynamic resource allocation requirements of a workload can be easily satisfied by resizing the capacity of a Virtual Machine (VM) at runtime. While, the ability to power-on/off, archive, migrate containers and their workloads enhance the capability of the resource managers to work around resource bottlenecks and faults. However, despite before mentioned capabilities, virtualization imposes a new challenge of added management complexity to the above-mentioned challenges of cloud platforms. So, it is still an open question of how to employ virtualization techniques effectively to realize a resource manager that efficiently and intelligently adapts the resource usage of cloud platforms to satisfy the conflicting objectives of the running applications and the underlying cloud infrastructures.

In an attempt to address the above-mentioned challenges, we have proposed in [2] a multi-level architecture that relays on a hybrid virtualization framework to provision resources by VMs instantiations, or horizontal scaling, fine grain tuning of resources by VMs resizing, or vertical scaling, and coarse grain tuning of the resources by live migration. In this paper, we propose some modifications to this model to reflect the progress of our ongoing investigation as shown in Fig. 1. Current model loosely couples decision-making process to decision implementation process. The process of decision making, or optimization, is completely shifted to the Local Configuration Planners (LCP), which operates at the granularity of the running applications. LCP has the duty of constructing configuration plans that maximize the utility of the running applications by tightly coupling three different aspects of the resource allocation problem of *what it is the optimal number, or optimal pattern, of VMs to be allocated for the running applications? What it is the optimal utilization levels to be assigned for the running VMs? What it the optimal allocation to be assigned for the running VMs?* On the other hand, Configuration Manger (CM) has a more general duty of monitoring and maintaining the optimum status for the running configuration within the virtualized environment as a whole by implementing a global reconfiguration plan based on the local ones, which constructed by LCPs. Resource allocation problem is an NP hard problem [3], and it is a good example of a heavy constrained dynamic optimization problem, especially if we consider the huge scale of the cloud platforms. So by aggregating the process of decision-making at a central module, we are at the risk of applying the scheduling algorithm to an extremely exhaustive problem that can consume an enlarged amount of time and resources (i.e. memory) to be addressed properly. Hence, the underlying decentralization in our model is of great advantage to support the scalable performance of the scheduling algorithm. On the other hand, the inherited parallelization within the model is another potential that can be utilized to enhance dramatically time performance of the optimization process to be compatible with the rapidly changed and dynamic nature of the resource allocation problem in cloud platforms.

On the other hand, Genetic Algorithms (GAs) are efficient search metaheuristics, which mimic natural

978-1-4673-1544-9/12/$31.00 ©2012 IEEE

evolution and have been applied successfully to a wide range of real-world applications. Recently, we have proposed Binary-Real coded GA (BRGA) [4] [5], which is a new hybrid approach that can handle a variety of binary and real coded optimization problems. BRGA relays on a parameterized hybrid scheme to share the computational power and coordinate the cooperation between a binary coded GA (BGA) and real coded GA (RGA) in an attempt to guide a population of candidate solutions through search space toward the optimum region of the optimization problems. BRGA has been applied successfully to a wide spectrum of global and constrained optimization problems from the known benchmark suites [6] [7] [8]. We have found that the performance of the BRGA is superior or comparable to the other state-of-the-art evolutionary algorithms. So in this paper, we aim to determine the mathematical model and present a modified version of BRGA to handle the proposed model. Moreover, we aim to evaluate the feasibility, effectiveness and scalability of our approach through simulation experiments.

The remainder of this paper is organized as follows. The position of this paper to the related literature is discussed in Section II. The proposed architecture is described, and the feasibility of its implementation is clarified in Section III. In Section IV, a formal mathematical model for resource allocation under the proposed approach is presented and discussed. A modified version of BRGA is presented in section V. while in Section VI, we report and discuss the outcomes of random dataset simulation experiments to evaluate the feasibility, effectiveness and scalability of the approach. Finally, the paper is concluded and possible directions of future work are highlighted in Section IV.

## II. LITERATURE REVIEW

Resource allocation in cloud platforms is an active research direction for both academic and industrial communities. Common cloud providers are in intense competition to develop competent resource management systems, which decides to a larger extend their abilities in slicing a greater share from a competitive cloud computing market with their state-of-the-art cloud computing services associated with advanced features. However, it is possible to argue that the literature is still lagging behind in revealing the underling secrets of those advance resource management systems In an attempt to minimize the gap, the main aim of our research is to realize an efficient and flexible resource management system by investigating new approaches in addressing open issues of resource management model, resource mapping formulation, scheduling algorithm and machine level tricks or techniques (i.e. virtualization).

Virtualization as a technology was developed originally in the 1960s to partition large, mainframe hardware for better hardware utilization. Virtualization was effectively abandoned in the 1980s and 1990s with the advent and popularity of the inexpensive personal computers [9]. However, the interest in virtualization was revived when it re-emerged as a flexible key technology to address the problems of modern distributed infrastructure like grid and cloud platforms. In a survey to the available literature [10], we have found that the majority of previous research considered either VMs live migration, VMs resizing or VMs instantiations as a single tool to manage their virtualized environment. This is especially suitable for smaller scale virtualized environments. Later, more complex hybrid virtualization frameworks were appeared that targeted grid and cloud platforms. For example, Ruth et al., in [11], presented a resource manager for a grid platform called Violin, which composed of virtual network of VMs. In Violin, the resource manager relays on a hybrid virtualization framework of VMs live migration and VMs resizing. On the other hand, Van et al., in [12], and later in [13], proposed an autonomic virtual resource manager that targets cloud platforms. The resource manager here relays on a hybrid platform consists from VMs instantiations, and VMs live migration. When compared with the above, the resource manager in our model utilizes a more diversified virtualization framework consists from VMs instantiations, or horizontal scaling, VMs resizing, or vertical scaling, and live migration. By adopting such virtualization framework, it becomes easier to utilize a new three-dimensional resource mapping formulation (status, utilization level and location) to address a larger set of control variables. Most importantly, utilizing such a hybrid framework can be an effective tool in itself to minimize virtualization overhead, which can be resulted from relaying on single virtualization technique.

From the aspect of underlying management model, virtual resource managers in the early period of research were heavily relayed on the single management architectures, which completely separate resource provisioning from resource management. An example is the resource manager proposed in [14] for small and medium IT environments, which controls resources through a coarse grain resource tuning (i.e. live migration). With the development of grid computing, researchers advocated more complicated management models. For example, Violin in [11] adopted a management model that loosely couples resource fine grain tuning (VMs resizing) with coarse grain tuning (VMs live migration). With the advent of cloud computing, "on-the-fly-provisioning" becomes more popular and researchers considered management models that loosely couples resource provisioning to resource management. An example is the manager in [12], where the management model loosely couples resource provisioning (VMs instantiations) to resource tuning (VMs live migration). When compared to the above, the management model in our proposed method is more diversified, since it integrates resource provisioning (VMs instantiations), fine grain resource tuning (VMs resizing) and coarse grain resource tuning (VMs live migration). By such extended model, we aim to address flexibly a wider range of the running applications and underlying infrastructures contradicting objectives. Moreover, our approach tightly couples the multi-level architecture at the LCPs level. By this kind of tightly coupling, we are hopeful that the optimization process is in a better position to capture optimal configurations by utilizing the potential synergism and/or picking up the right existed trade-offs among the multi-dimensional resource allocation model.

Finally, GAs have been applied successfully to a variety of complex real-world applications. The efficiency of GAs in exploiting modern parallelization paradigms [15] make them a potential candidate to address complex resource allocation problems in huge scale of clouds. However, according to [10], ranking heuristic procedures is still more popular for virtual resource managers. Some of the few applications of GAs are [16] and [17]. In [16], authors used a binary-coded GA for online self-reconfiguration in large-scale cloud computing data centers. On the other hand, in [17] a real-coded GA was used to handle virtual Machine packing optimization problem on Grivon, which is a virtual cluster management system for Hardware as a Service (HaaS) cloud. Compared with the above, BRGA is a recent hybrid approach, and it can address simultaneously binary and real coded decision variables. So, BRGA is a suitable candidate to handle multi-dimensional resource mapping formulation in our approach. Most importantly, BRGA is robust in the sense that does not require any sophisticated and expensive constrains repairing procedures or to incorporate special techniques to handle the proposed approach effectively.

## III. System Architecture

An overview of the proposed architecture is shown in Fig 1. Following the design principles of SOA, each application is encapsulated by an Application Environment (AE), which decouples it from the underlying physical infrastructure. AE, as an embedded technology, realizes the key characteristics of compatibility, isolation, encapsulation, hardware independence and improved manageability. Following the practices of common cloud providers, like Amazon EC2, AE under our model can rely on virtual appliances. According to [18], virtual appliances are self-contained and optimized application stacks that are customized for their workload and embedded with an Operating System of choice. Virtual appliances can be an effective tool for enhancing security, isolation and reliability within the virtualized environment. Most importantly, when compared with traditional approaches for resource provisioning, virtual appliances can effectively eliminate complex manual provisioning steps, which can consume time and labor. In other words, virtual appliances are more compatible with the "on-the-fly-provisioning" paradigm of cloud platforms. In the remainder of the paper, we assume that each virtual appliance is implemented using a single VM. As a terminology, both virtual appliances and VMs can be used interchangeably. So by Virtualized Application Environments (VAEs), we refer to groups of VMs which serve users' specific running applications. The number of VMs within a specific VAE, at a specific control period, can vary within a range limited by upper and lower bounds, which are defined by SLA. Moreover, VMs instances within VAEs follow standard computational resource specifications defined by instant types available from the cloud provider, e.g. see [19]. On another hand, there is a Local Agent (LA) attached with each VAE. LA is an intelligent software component which has the duty of maintaining the QoS requirements of the VAE by producing estimates for the requested resources during the next control interval. It maintains a performance model which relates SLA's quantitative metrics (e.g. throughput and latency) with the consumed computational resources. The performance model can be analytical [20] or empirical [12]. Through the monitoring probes, LA continuously collects data about the status of the running VMs and the underlying physical machines (PMs) and saves them in the repository. A sliding window of this historical data can be used to forecast demand estimates for the next control interval, e.g. see [21].

There is a LCP attached to each VAE. LCP has the duty of constructing a reconfiguration plan from a three-dimensional resource mapping formulation which maximizes active VAEs local utility function. LCP takes workload estimates from LA, while it takes pre-processing information from the CM. LCP starts optimization whenever it initialized by CM. On the other hand, CM maintains a general view of the managed environment through the monitoring probes attached to the running VMs and the underlying PMs. It has the duty of acting proactively against system failure by analyzing PMs level health data (i.e. temperature, fan speed), e.g. see [22]. Moreover, it maintains

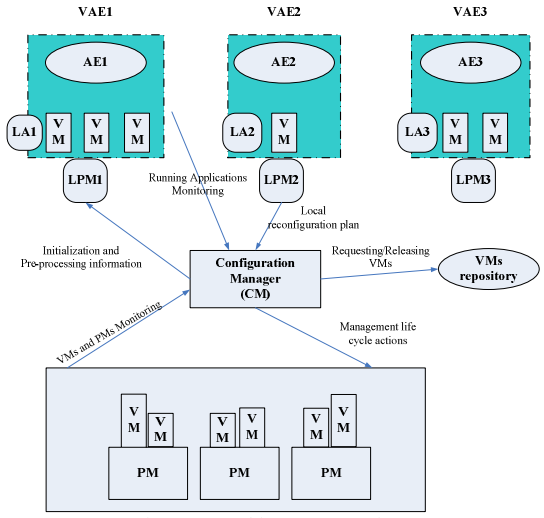


Figure 1. Architecture overview.

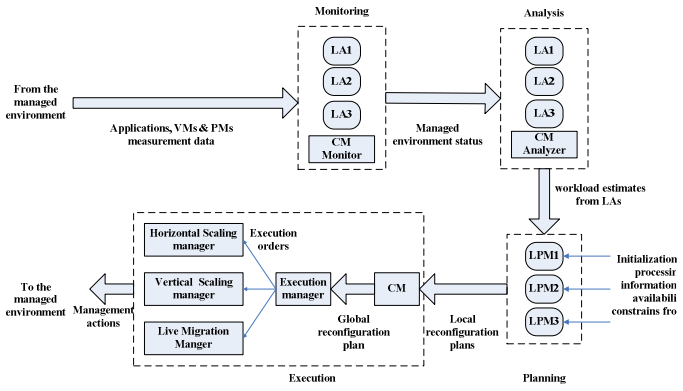


Figure 2. Autonomic management loop.

management system metrics like the duration of control interval and communicates pre-processing information (i.e. like the status of the current configuration, the health status of the PMs and defines VAE budget from the physical resources). Finally, CM has the responsibility of maintaining the optimum status of the running configuration within the virtualized environment by taking the appropriate VMs lifecycle management actions based on the local reconfiguration plans constructed by the LCPs.

Moreover, the resource manager adopts a common Monitoring-Analysis-Planning-Execution or MAPE autonomic management loop, which is processed within a specific control interval. The management loop is shown in Fig. 2. In the monitoring phase, CM keeps a global view on the management environment by collecting measurement data from the running VMs and their underlying PMs, and active PMs health data. In addition, it tracks updates done by the users on the metrics of the running VAEs (i.e. the VMs class size). On other hand, LAs keeps a local view by collecting measurement data from their VAEs running VMs and underlying PMs. In the analysis phase, LAs process the collected measurement data to capture workload variability trends of their VAEs and produce workload estimates for the next control interval. In addition, CM's analyzer processes the collected active PMs health data to update the availability flags which enable CM to guarantee service availability by reacting proactively to any possible hardware failures through live migration from unhealthy nodes to the healthy ones. After constructing the optimal reconfiguration plans by LCPs in the planning phase, it is the duty of the CM to regulate the active configuration within the managed environment to the new optimal status though the Execution manager in the Execution phase. The Execution manager compares the current configuration by the new optimal one to initiate suitable orders, which implemented by a hybrid virtualization framework consist from horizontal scaling manager, vertical scaling manager and live migration manager. Horizontal scaling manager implements orders of requesting/releasing VMs. While the vertical scaling manager employs memory ballooning and CPU scheduling techniques to implement orders of scale up or scale down the utilization levels of the running VMs, e.g. see [11]. Moreover, migration manager implements orders of VMs live migration, PMs power on or PMs power off.

Finally, Table I summarize the main features of the proposed resource manager. It clarifies how these features can serve the overall objectives of the resource manager. On the other hand, the analysis in the remainder of paper will be limited to planning phase due to the importance of this part, which represents the core of the resource manager intelligence.

## IV. MATHEMATICAL MODEL

In this section, we formulate the resource allocation problem in cloud platforms under the proposed management model as a mixed integer optimization problem. The list of notations is clarified in Table II.

We assume that the platform can host up to M running applications or VAEs. The platform is consisting from P PMs and N VMs, which are following K instant types. At the time being, we limit our analysis to the processing power (CPU) and memory (RAM) as most important and competing for resources. So, both hardware and computational capacity within the platform can be described as follows.

$$p_i = [p_i^{CPU}, p_i^{RAM}] \; and \; Pc = \sum_{i=1}^{P} p_i \quad (1).$$

$$v_{i,j,k} = [\, v_{i,j,k}^{CPU}, v_{i,j,k}^{RAM} \,] \; and \; \forall j,k \;\; V_c = \sum_{i=1}^{N} v_{i,j,k} \quad (2).$$

On the other hand, the computational power can be expressed in terms VAEs as follows.

$$V_c = \sum_{j=1}^{M} V_j^{class} \;, \forall j,k \;\; V_j^{class} = \sum_{i=1}^{nmax_j} v_{i,j,k}$$

$$and \; N = \sum_{j=1}^{M} nmax_j \quad (3).$$

Where, $nmax_j$ is usually decided by the SLA. It should be noted that under the proposed model, we assume that the dynamic resource manager is responsible on respecting SLA's target by provisioning resource up to $nmax_j$.

On the other hand, capacity relationship between Pc and Vc are planned such that additional PMs are provisioned as a reserve to resolve critical situation of sudden drop out of some of the active PMs from the platform due to system failure without affecting end users QoS such that;

$$Pc = Vc + \Theta_{rese} * Vc \;\; \text{and} \;\; 0 < \Theta_{rese} \leq Vc \quad (4).$$

Where $\Theta_{rese}$, which is maintained by CM, can be flexibly controlled by the administrator to manage system preparedness against critical issues of maintenance and

TABLE I THE PORPOSED ARCHITECURE OBJECTIVES AND FEATURES

| Objective | Features | Remarks |
|---|---|---|
| Maximize user benefit | QoS aware | The self-optimization architecture is guided by QoS requirements, which expressed in terms of quantitative metrics. |
| | Service availability guaranteeing | The system is designed to respond proactively to expected system failures by quickly migrate VMs from unhealthy nodes to the healthy ones. |
| | Elasticity | The system is designed to add or subtract resources from the running applications seamlessly. That maximizes users` benefit under pay-as-you-go model. |
| Minimize System overhead | Eliminating under utilization | The system is design to detect and eliminate underutilization. That contributes effectively in reducing system operation costs and limiting system ecological effects. |
| Minimize Virtualization overhead | On-the-fly provisioning | Horizontal scaling of pre-configured VMs will be faster and eliminate the complex steps and time required for server provisioning. |
| | Minimize migration overhead | Migration overhead is addressed by effective use of vertical scaling and horizontal scaling. Also, Migration manager is designed to minimize the number of steps required for VMs migration. |

system failure.

So by taking the above into consideration, it is possible to state that the resource manager describes the configuration status within the platforms using a global vector which contains three sub-vectors of $V_{status}$, $V_{frac}$ and $V_{place}$ as follows.

$$V_{confg} = [V_{status}, V_{fac}, V_{place}] \quad (5).$$

Where each sub-vectors corresponds to a specific management level.

$V_{status}$ is used by the horizontal scaling manager to optimally scale, upwardly or downwardly, the number of actives VMs within a specific VAE. $V_{status}$ can be described as follows.

$$\forall j,k\ V_{status} = [vs_{1,j,k}, vs_{2,j,k}, \dots vs_{i,j,k}, \dots, vs_{N,j,k}]$$

$$\text{Where } vs_{i,j,k} = \begin{cases} 1 & \text{if } v_{i,j,k} \text{ is active} \\ 0 & \text{otherwise} \end{cases} \quad (6).$$

Moreover, $V_{frac}$ is used by the vertical scaling manager to assign an optimal utilization level for the running VMs within the managed environment. $V_{frac}$ can be described as follows.

$$\forall j,k\ V_{frac} = [vf_{1,j,k}, vf_{2,j,k}, \dots vf_{i,j,k}, \dots, vf_{N,j,k}],$$

$$\text{where } 0 < vf_{i,j,k} \leq 1 \quad (7).$$

Finally, $V_{place}$ is used by the live migration manager to assign an optimal placement for the running VMs on the underlying active PMs. $V_{place}$ can be described as follows.

$$\forall j,k\ V_{place} = [vp_{1,j,k}, vp_{2,j,k}, \dots, vp_{i,j,k}, \dots, vp_{N,j,k}],$$

$$\text{Where } vp_{i,j,k} \in [1, P] \quad (8).$$

So,$U_{j1}$ is formulated to tune finely the provisioned resource onto the requested demand. Following the practices of the common cloud provider, the resources are usually provisioned to the users in the form of discrete bulks of VMs. However, VAEs actually consumes these bulks by fractions. So, by detecting and eliminating the existed gaps between the provisioned resource and requested demand, resource manager can save energy by increasing the compactness of the reconfiguration plan. Regulation $U_{j1}$ is done by considering constraints (10) and (11) to avoid overutilization and violation of SLA's targets.

Moreover, $U_{j2}$ is formulated to minimize system overhead by detecting and minimizing migration overhead. Initiating of excessive live migrations can be expensive, especially in huge platforms like clouds. So, $U_{j2}$ assigns better scores for configurations as near as possible from the current status within the managed environment. In other words, it detects configurations, which generate fewer live migrations. Regulation of $U_{j2}$ is done by considering constraint (13) to guarantee system availability by quickly migrate VMs from unhealthy nodes to the healthy ones.

In addition, $U_{j3}$ is formulated to detect the correct reserved instance pattern that can afford the requested demand. According to pay-as-you-go paradigm, users pay more for instances of higher computational capacity type than those of lower computational capacity type [19]. Failing in identifying the right pattern can damage the reputation of the cloud provide, since it violates the above-mentioned paradigm. So, $U_{j3}$ assigns better scores to configurations that minimize the gap, as much as possible, between the requested demand and the active VMs pattern

Table II. LIST OF NOTATIONS

| | |
|---|---|
| M | Number of the running VAEs. |
| N | Total number of the VMs within the managed environment. |
| P | The total number of the PMs within the managed environment. |
| K | The total number of the instant types within the managed environment. |
| t | An integer valued number refers to the current discretized control interval. |
| t+1 | An integer valued number refers to the next discretized control interval. |
| $\alpha_{SLA}$ | A faction refers to SLA performance targets in terms of the resources |
| L | The number of active VMS assigned to a specific PM. |
| $P_i$ | Hardware capacity of *i*th PM within the managed environment. |
| $P_i^{CPU}$ | Hardware capacity of ith PM in terms of processing power, (CPU ). |
| $P_i^{RAM}$ | Hardware capacity of ith PM in terms of memory, (RAM). |
| Pc | Total hardware capacity within the managed environment. |
| $\Theta_{rese}$ | Fraction from the hardware budget of *i*th VAE reserved for to guarantee service availability. |
| $\theta_i$ | Fraction of ith PM hardware capacity reserved to compensate the effects of virtualization overhead. |
| $v_{i,j,k}$ | The standard computational capacity of ith VM within jth VAE class of kth instant type. |
| $v_{i,j,k}^{CPU}$ | The standard computational capacity of ith VM within jth VAE class of kth instant type, in terms of processing power (CPU). |
| $v_{i,j,k}^{RAM}$ | The standard computational capacity of ith VM within jth VAE class of kth instant type, in terms of memory (RAM). |
| Vc | The total computational power reserved for the running VAEs within the managed environment. |
| $V_j^{class}$ | The share of jth VAE from Vc in terms of computational power. |
| $nmax_j$ | The maximum number of VMs reserved for jth VAE within the managed environment. |
| $nmin_j$ | The minimum number of VMs reserved for jth VAE within the managed environment. |
| $n_j^t$ | The number of active VMs which serves jth VAE at t. |
| $Vr_i^{t+1}$ | The requested workload demand for ith VAE class at next control interval. |
| $V_{confg}^t$ | The status of the global configuration within the managed environment at t. |
| $V_{status}^t$ | A Boolean vector refers to the activation status of VMs within the managed environment at t. |
| $vs_{i,j,k}$ | A Boolean valued number refers to the activation status of ith VM within jth VAE class of kth type . |
| $V_{frac}^t$ | A real valued vector refers to the utilization level of VMs within the managed environment at t. |
| $vf_{i,j,k}$ | A real valued number refers to the utilization level of VMs within the managed environment. |
| $V_{place}^t$ | An integer valued vector refers to the placement of the VMs on the underlying PMs within the managed environment at t. |
| $vp_{i,j,k}$ | An integer valued number refers to the placement of ith VM within jth VAE class of kth type on the pth PM. |
| F | A Boolean vector refers to the availability flags. |
| $U_g$ | Global utility function maintained by CM |
| Uj | jth VAE local utility function maintained by LCP |
| w | Tradeoff weights used to attached importance to a specific objective (i.e. local utility function) |

for a specific VAE. Regulation of is done by considering constraints (10) and (11) to avoid overutilization which can affect QoS requirements.

On other hands, $U_{j4}$ is formulated to minimize energy consumption within the managed environment by minimize the number of active PMs. So, $U_{j4}$ eliminates underutilization by increasing the compactness of the running VMs. Regulation of $U_{j4}$ is done by considering constraints (12) which prevent overutilization that can affect QoS. By utilizing the potential synergism between $U_{j4}$ and $U_{j1}$, the scheduling algorithm can identify better configurations from the compactness stand point. On the hand, by picking up the trade-off between $U_{j4}$ and $U_{j2}$, the scheduling algorithm can avoid configurations, which generate wasteful migration steps, i.e. migrations, which do not increase the compactness of the managed environment.

Finally, $U_{j5}$ is formulated to minimize the system overhead by minimizing VMs activations/de-activation. So, $U_{j5}$ assigns better scores for configurations as near as possible from the current status within the managed environment. In other words, it detects configurations, which use less activation/deactivation steps. By picking up the trade-off between $U_{j5}$ and $U_{j3}$, the scheduling algorithm can avoid configurations, which use wasteful activation/deactivation steps. Specifically, it avoids activation/deactivation steps that do not improve the quality of the active VMs patterns. Regulation of $U_{j5}$ is done by considering constraints (10) and (11) to avoid overutilization which can affect QoS requirements.

## V. OPTIMIZATION ALGORITHM

In this section, we refer to the optimization algorithm, i.e. BRGA. However, due to the page limitations, we will avoid the lengthy discussion, since it can be found in our previous work. Rather, we emphasize briefly the principal modifications which make BRGA compatible to handle the proposed approach effectively. The main parts of the algorithm can be explained as follows.

(1). **Chromosome representations**: a floating point and a binary representation were used to encode $V_{confg}$ These populations are kept updating by continuously mapping from one version to another during the evolutionary cycle. In the binary version chromosomes, both $V_{frac}$ and $V_{place}$ are mapped into the equivalent binary values. While, in the real version chromosomes only $V_{status}$ sub-vector is mapped into the equivalent real value.

(2) **Binary coded GA**: The main duty of BGA is search space explorations to abstract promising regions within it. This part is implemented using a binary GA as described by Haupt and Haupt [23] . The main parts of this algorithm are single point crossover operator, one-point crossover operator, rank weighting selection scheme However, the crossover operator, here, is modified to respect the boundary of the sub-vectors of $V_{confg}$ .

(3) **Real coded GA:** the role of RGA is to adapt the population toward optimality region by extensively exploit promising regions within search space, which identified by BGA. This part is implemented using Unimodal Normal Distribution (UNDX) as an advanced real crossover operator [24] and Minimal Generation Gap (MGG) is used as a generation-alteration model. The main modification here is chromosome representation, since it uses two representations. One is similar to $V_{confg}$, which is used for population evaluation. In the other one, the $V_{status}$ sub-vector is mapped into the equivalent real value. This representation is used for population processing.

(4) **Population handover:** it is a recent adaptive parameter based hybrid scheme. Population handover

$$\text{Max}\,(U_g) = \text{Min}\,((w1 * U_{j1} + w2 * U_{j2} + w3 * U_{j3} + w4 * U_{j4} + w5 * U_{j5})/M)$$

Where:

$$\forall j,k \quad U_{j1} = \frac{\sum_{i=1}^{N} vs_{i,j,k}^{t+1} * vf_{i,j,k}^{t+1} * v_{i,j,k}}{Vr_i^{t+1}}$$

$$\forall j,k \quad U_{j2} = \frac{\sum_{i=1}^{N} F\left(\left|vp_{i,j,k}^{t+1} - vp_{i,j,k}^{t}\right|\right) * vs_{i,j,k}^{t+1} * vs_{i,j,k}^{t}}{\sum_{i=1}^{N} vs_{i,j,k}^{t+1}}$$

$$\text{and } \forall i,j,k \quad F\left(\left|vp_{i,j,k}^{t+1} - vp_{i,j,k}^{t}\right|\right) = \begin{cases} 1 \text{ if } \left|vp_{i,j,k}^{t+1} - vp_{i,j,k}^{t}\right| \neq 0 \\ 0 \quad \text{otherwise} \end{cases}$$

$$\forall j,k \quad U_{j3} = \frac{\sum_{i=1}^{N} vs_{i,j,k}^{t+1} * v_{i,j,k}}{Vr_j^{t+1}}$$

$$\forall j \quad U_{j4} = \frac{\sum_{i=1}^{P} F(p_i)}{Vr_j^{t+1}} \text{ and } \forall i,j \;\; F(p_i) = \begin{cases} Pi \quad \text{if ith PM} \in V_{place}^{t+1} \\ 0 \quad \text{otherwise} \end{cases}$$

$$\forall j,k \;\; U_{j5} = \frac{\sum_{i=1}^{N} \left|(vs_{i,j,k}^{t+1} - vs_{i,j,k}^{t})\right|}{\sum_{i=1}^{N} vs_{i,j,k}^{t+1}} \qquad (9).$$

Subject to:

$$\forall j \;\; nmax_j \leq n_j^{t+1} \leq nmax_j \qquad (10).$$

$$\forall j,k \;\; \sum_{i=1}^{N} vs_{i,j,k}^{t+1} * vf_{i,j,k}^{t+1} * v_{i,j,k} \geq Vr_i^{t+1} \qquad (11).$$

$$\forall p_i \in V_{place}^{t+1}, vp_{i,j,k}^{t+1} \in p \;\; \sum_{i=1}^{L} vs_{i,j,k}^{t+1} * vf_{i,j,k}^{t+1} * v_{i,j,k} \leq \alpha_{SLA} * p_i - \sum_{i=1}^{L} \theta_i \qquad (12).$$

$$\forall p \in V_{place}^{t+1} \;\; F_p = 1, \text{ Where } F = [F_1, F_2, \ldots, F_p, \ldots, F_P] \text{ and } F_p = \begin{cases} 0 \text{ if p is Unhealthy} \\ 1 \quad \text{otherwise.} \end{cases} \qquad (13).$$

Figure 3. Resource Allocation Mathematical Model.

coordinates the cooperation between BGA and RGA by a frequent process of population mapping from the BGA to RGA and vice versa. By controlling the frequency and the amount of data exchange through population handover; BRGA adopts a strategy of consecutively expanding-intensifying the evolutionary search in an attempt to guide population of solutions rapidly and successfully toward optimality regions.

(5) **Dynamic constraint-handling technique**: Dynamic constraint-handling technique is a category of penalty functions in which generation number is involved in the computation of the corresponding penalty factors. This part is implemented using a modified version for a dynamic technique proposed by Joines and Houck [25], which can be defined as follows for minimization problems.

$$\hat{f}(X) = f(X) + (C * t)^{\alpha} * SVC(\beta, X) \quad (14).$$

$$SVC(\beta, X) = \sum_{i=1}^{p} G_i^{\beta 1}(X) + \sum_{j=p+1}^{m} H_j^{\beta 2}(X) \quad (15).$$

Where, $f(X)$ is the original function, $\hat{f}(X)$ is penalized function, Gi is inequality constraints, Hj is equality constraints, m is the total number of constraints, t is generation number, while C, α, β1, β2 are penalty factors. The value of β1 equals β2 and can be defined as follows.

$$\beta 1 = \begin{cases} \beta 1 & \text{if } v_1 > 1 \\ 1 & \text{if } v_1 \leq 1 \end{cases}, \beta 2 = \begin{cases} \beta 2 & \text{if } v_2 > 1 \\ 1 & \text{if } v_2 \leq 1 \end{cases} \quad (16).$$

Where:

$$v_1 = \sum_{i=1}^{p} G_i(X), v_2 = \sum_{j=p+1}^{m} H_j(X)$$

$$G_i(X) = \begin{cases} g_i(X) & \text{if } g_i(X) > 0 \\ 0 & \text{if } g_i(X) \leq 0 \end{cases}$$

$$H_i(X) = \begin{cases} |h_j(X)| & \text{if } |h_j(X)| - \varepsilon > 0 \\ 0 & \text{if } |h_j(X)| - \varepsilon \leq 0 \end{cases} \quad (18).$$

Where ε is resolution and it is set in this paper as (0.001).

## VI. Experimental Dataset and Evaluation

In this section, we aim to evaluate the feasibility, effectiveness and scalability of the proposed approach by reporting and discussing the outcomes of simulation experiments. One of the major difficulties which faces the research within the field cloud computing is the absence of publicly known datasets which depicts the structures of the virtualized environments within cloud platforms. With the absence of such of dataset, it is difficult to comprehend comprehensively the effectiveness of the scheduling algorithms and comparing against those developed by other researchers. However, to overcome this difficulty we considered in our experiments a group of randomly generated configurations that depicts specific aspects to challenge the performance of the algorithm.

The reported results in this section are based on 25 experimental runs. The length of single run is 10,000 generations. We assumed that SLA's requirements are strict, 99%, while virtualization overhead is small,(θi=0.01) Moreover, we assumed that all PMs are under normal, healthy, condition, i.e. the flags of F vector are 0. The PMs in our environments is homogeneous, each with 4 MHz CPU and 4 MGB RAM. The VMs in our environments are of four types small (0.5 MHz CPU, 0.5 MGB RAM), medium (1 MHz CPU, 1 MGB RAM), large (2 MHz CPU, 2 MGB RAM) and extra large (3 MHz CPU, 3 MGB RAM). In all the experiments, a fixed parameterization derived from our previous work was used for BRGA. So, The considered experimental scenarios can be described as follows.

Table IIII. FEASIBILTY EXPERIMENTS AT C1

| status at $V_{cong}^{t}$ | Fitness | PMS | VMs | C.R. | R.R. |
|---|---|---|---|---|---|
| | 121.19 | 2 | 4 | 26% | 46% |
| status at $V_{cong}^{t+1}$ | **Fitness** | **Min** | **Mean** | **Max** | **Success rate** |
| | **Ug** | 5 | 5.04 | 5.65 | 92% |
| | **$U_{11}$** | 1 | 1 | 1 | 100% |
| | **$U_{12}$** | 0 | 0.01 | 0.25 | 96% |
| | **$U_{13}$** | 1 | 1.01 | 1.15 | 96% |
| | **$U_{14}$** | 1 | 1 | 1 | 100% |
| | **$U_{15}$** | 0 | 0.02 | 0.5 | 96% |

A. **Feasibility experiments**: Since LCP is operating at the granularity of individual VEAs. So, here we limit the analysis to the level of single VAE. The status of the simulated environment is summarized in Table III. We assumed that our environment is consisting from 5 PMs and 10 VMs which are serving a single VAE. The reserved VMs have a capacity pattern of VMs as 2 extra large, 2 large, 3 medium and 3 small VMs. The active configuration, C1, contains 4 active VMs and 2 active PMs. C1 consumes 26% of the reserved budget (C.R.), while the requested demand requires 45% of the reserved budget (R.R.). Same importance, (trade-off weights equal 1), were attached to the utility functions. By the best-case scenario analysis, we found that QoS can be met by adjusting upwardly the utilization levels of the current active VMs. So, C1 is useful configuration to reveal how smart BRGA is in constructing the optimal reconfiguration plan. Since, the best-case scenario does not require live migrations, VMs activations/deactivations and PMs activation/deactivation.

Table III shows that BRGA was successful in picking up the optimal configuration with success rate of 92%. BRGA had a perfect performance in addressing the requirements of energy ($U_{14}$ and $U_{11}$), while it achieved a high score (96%) in addressing the requirements of user satisfactions ($U_{13}$) and system overhead minimization ($U_{12}$ and $U_{15}$).

**B. Variable workload intensity experiments**: In this phase, we aim to calrify the performance of BRGA against workload of different intensities. Again, we limit the analysis here to the level of single VAE. Table IV summarized the situation within the platform at configurations with different

workload intensities. We assumed that our environment is consisting from 5 PMs and 10 VMs which are serving a single VAE. The reserved VMs have a capacity pattern of 2 extra large, 2 large, 2 medium and 4 small VMs. Higher importance has been attached to $U_{j3}$, (w3 = 1.75), while lower importance has been attached to $U_{j7}$, (w5 = 0.25). The same importance (weights equal one) has been attached to the remaining utility functions. We have 6 configurations, (C2-C7). The first three simulates the situation of overutilization managed environment, while the last three simulate the situation of underutilization managed environments with different workload intensities. To test the ability of BRGA in adapting the number of active VMs, all of the reserved instances were activated. On the other hand, to examine the ability of BRGA in adapting the number of active PMs, only one of the PMs have been activated in case of overutilization scenario, while all of the PMs have been activated in the case of underutilization.

Table IV shows that BRGA was successful in satisfying the requirements of QoS with a perfect feasibility rate (F.R.). On the other hand, the small values of standard deviations reflect that BRGA was successful in maintaining a robust performance against the variability in workload intensities.

**C. Scalability experiments**: In this phase, we aim to test the performance of BRGA against larger-scale environments. We have considered two scenarios of (5 VAES, 50 VMs, 25 PMs), (10 VAES, 100 VMs, 50 PMs), (50 VAES, 500 VMs, 250 PMs) and (100 VAES, 1000 VMs, 500 PMs) environments. Capacity pattern and weights for the VAES in both cases are similar to those in previous step. At the current configurations, all the VMs were activated to stress BRGA ability on adapting them. Random number of PMs was activated to simulate average case scenarios. Requested demands were generated to simulate a moderately overutilization environment scenario.

Fig. 4 depicts the situation at the current randomly generated configurations, while Fig. 5 reports the performance of BRGA at the optimal obtained ones. Again, BRGA was successful in satisfying the requirement of QoS with perfect feasibility rate and smaller values for the final fitness. Furthermore, BRGA was successful in maintaining a robust performance against environments of larger running applications and computational resources.

## VII. Conclusion and Future Work

Resource allocation in cloud platforms is a dynamic heavy constrained problem which poses multi facet challenges against the development of efficient and reliable management systems. In an attempt to realize an intelligent resource manager which can flexibly address the contradicting objectives of the running applications and the underlying infrastructure, we have proposed a modified multi-level autonomic resource manager. Furthermore, we have formulated a formal mathematical model and proposed a modified BRGA to handle the architecture effectively. Simulation experiments based on randomly generated datasets of specific challenging aspects under different scenarios were conducted. The outcomes confirm the feasibility, effectiveness and scalability of the approach. Moreover, the experiments reflect that BRGA has stable performance in resolving the problem of resource allocation under the proposed approach.

On the other hand, possible directions for future work include conducting of test bed experiments to analyze the performance of the proposed approach under critical dynamic scenarios, expanding the mathematical model with empirical models, investigating the possibility of adding second round of optimization at CM to increase the compactness of reconfiguration plans, and finally employing modern parallelization paradigm to build faster, and stronger version of BRGA based resource manager.

## ACKNOWLEDGMENT

We would like to express our sincere gratitude to the anonymous reviewers for their efforts and comments, which help considerably in improving the quality of this paper. Moreover, this work is supported by KAKENHI (No.22500196), Japan Society for the Promotion of Science.

## References

[1] B. Addis, D. Ardagna, B. Panicucci, Li Zhang. Autonomic Management of Cloud Service Centers with Availability Guarantees. In Proc. of 2010 IEEE 3rd International Conference on Cloud Computing, Miami, FL, USA, 5-10 July 2010, pp. 220 – 227.

[2] O. Abdul-Rahman, M. Munetomo, K. Akama. Multi-Level Autonomic Architecture for the Management of Virtualized Application Environments in Cloud Platforms. In Proc. of 2011 IEEE 4th International Conference on Cloud Computing, Washington DC, USA, 4-9 July 2011, pp. 754 - 755.

[3] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici. A scalable application placement controller for enterprise data centers. In WWW2007, 2007.

[4] O. Abdul-Rahman, M. Munetomo and K. Akama. An Adaptive Resolution Hybrid Binary-Real Coded Genetic Algorithm. In Proc.of the 16th International Symposium on Artificial Life and Robotics (AROB 16th '11), Jan. 2011, Japan, pp. 359-362.

[5] O. Abdul-Rahman, M. Munetomo and K. Akama. An Adaptive Resolution Hybrid Binary-Real Coded Genetic Algorithm. Journal of Artificial Life and Robotics, Springer, Vol. 16(1), 2011, pp. 121-124.

[6] O. Abdul-Rahman, M. Munetomo and K. Akama. An Improved Binary-Real Coded Genetic Algorithm for Real Parameter Optimization. In the Proc. of 2011 3rd Congress on Nature and Biologically Inspired Computing (NaBIC), Salamanca, Spain, 19-21 Oct. 2011, pp. 149 – 156.

[7] O. Abdul-Rahman, M. Munetomo and K. Akama. An Adaptive Parameters Binary-Real Coded Genetic Algorithm for Real Parameter Optimization: Performance Analysis and Estimation Of Optimal Control Parameters. International Journal of Computer Science Issues (IJCSI), Vol. 9(2), March 2012, *in press*.

[8] O. Abdul-Rahman, M. Munetomo and K. Akama. An adaptive Parameters Binary Real coded Genetic Algorithm for Constraint Optimization Problems: Performance Analysis and Estimation of Optimal Control Parameter. Journal of Information Sciences, Elsevier, *unpublished*.

[9] VMware. http://www.vmware.com/virtualization/history.html

[10] O. Abdul-Rahman, M. Munetomo and K. Akama. Live Migration based Resource Managers for Virtualized Environments: A Survey. In the Proc. of the 1st International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2010), Lisbon, Portugal, Nov. 2010, pp. 32-40.

[11] P. Ruth, J. Rhee, D. Xu, R. Kennell, and S. Goasguen. Autonomic Live Adaptation of Virtual Computational Environments in a Multi-

TABLE IIIV. VARIABLE WORKLOAD INTENSITIES EXPERIMENTS AT (C2-C7)

| Config. | | Status at Vconfigt | | | | | Status at Vconfigt+1 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Fitness | PMs | VMs | C. R | R.R | Fitness | | | | F.R. |
| | | | | | | | Min | Mean | Max | std | |
| overutilization | C2 | 2.68E+02 | 1 | 10 | < 25% | (50-25)% | 6.14 | 6.33 | 6.69 | 1.45E-01 | 100% |
| | C3 | 3.97E+02 | 1 | 10 | < 50% | (75-50)% | 6.20 | 6.38 | 6.63 | 9.13E-02 | 100% |
| | C4 | 1.00E+03 | 1 | 10 | < 75% | (100-75)% | 6.26 | 6.46 | 6.98 | 1.33E-01 | 100% |
| underutilization | C5 | 2.60E+01 | 5 | 10 | (100-75)% | < 75% | 6.16 | 6.35 | 6.49 | 8.92E-02 | 100% |
| | C6 | 1.64E+01 | 5 | 10 | (75-50)% | < 50% | 6.52 | 6.53 | 6.74 | 4.49E-02 | 100% |
| | C7 | 1.42E+01 | 5 | 10 | (50-25)% | < 25% | 6.52 | 6.51 | 6.64 | 6.64E-02 | 100% |


Domain Infrastructure. In the Proc. IEEE ICAC '06, June, 2006, pp. 5 – 14.

[12] H. Van and J. Menaud. Autonomic Virtual Resource Management for Service Hosting Platforms. In the Proc. of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing, 2009, pp. 1-8.

[13] Hien Nguyen Van, F.D. Tran, J. M. Menaud. Performance and Power Management for Cloud Infrastructures. In Proc. of 2010 IEEE 3rd International Conference on Cloud Computing, Miami, FL, USA, 5-10 July 2010, pp. 329 – 336.

[14] G. Khanna, K. Beaty, G. Kar, A. Kochut. Application Performance Management in Virtualized Server Environments. Network Operations and Management Symposium 2006 NOMS 2006 10th IEEEIFIP, April, 2006, pp. 373 – 381,

[15] A. Munawar, M. Munetomo and K. Akama. Redesigning Evolutionary Algorithms for Many-Core Processors. PhD. Thesis, Hokkaido University, Japan, 2011.

[16] Mi Haibo et. al. Online Self-Reconfiguration with Performance Guarantee for Energy-Efficient Large-Scale Cloud Computing Data Centers. In Proc. of 2010 IEEE 3rd International Conference on Cloud Computing, Miami, FL, USA, 5-10 July 2010, pp. 514 – 521.

[17] H. Nakada, T. Hirofuchi, H. Ogawa and S. Itoh. Toward Virtual Machine Packing Optimization Based on Genetic Algorithm. In the Proc. of the 10th International Work-Conference on Artificial Neural Networks: Part II: Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living, Salamanca, Spain, 2009, pp. 651 – 654.

[18] VMware, virtual appliances. http://www.vmware.com/technical-resources/virtualization-topics/virtual-appliances/faqs

[19] Amazon web services. http://aws.amazon.com/ec2/pricing/

[20] B. Abrahao, V. Almeida, J. Almeida, A. Zhang, D. Beyer, and F. Safai. Self-Adaptive SLA-Driven Capacity Management for Internet Services. In Proc. NOMS06, 2006.

[21] N. Bobroff, A. Kochut and K. Beaty. Dynamic Placement of Virtual Machines for Managing SLA Violations. IEEE/IFIP International Symposium on Integrated Network Management (IM), Munich, Germany, May 21-25, 2007, pp. 119 – 128.

[22] A. B. Nagarajan, F. Mueller, Ch. Engelmann, S. L. Scott. Proactive Fault Tolerance for HPC with Xen Virtualization. In the Proc. of the 21st annual international conference on Supercomputing (ICS '07), New York, NY, USA, 2007.

[23] R. Haupt and S. Haupt. Practical Genetic Algorithms. Wiley-Interscience,NewYork, 1998.

[24] I. Ono and S. Kobayashi. A Real-Coded Genetic Algorithm for Function Optimization Using Unimodal Normal Distribution Crossover. In the Proc. of the 7th ICGA, 1997, pp 246–253.

[25] J.A. Joines and C.R. Houck. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GA's. In the Proc. of the First IEEE World Congress on Computational Intelligence, 2 (1994), pp. 579-584.


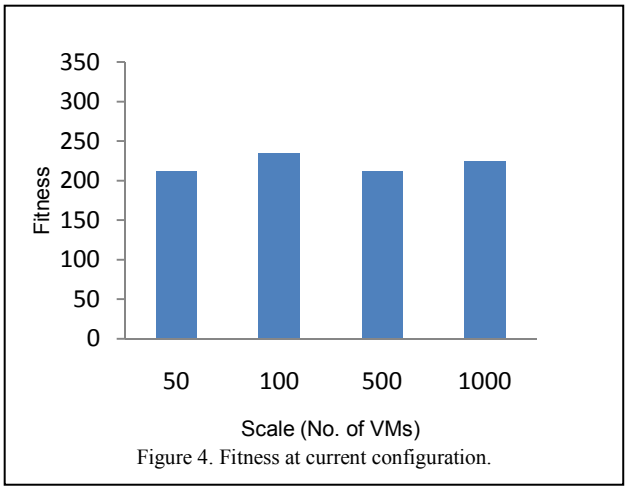


Figure 4. Fitness at current configuration.

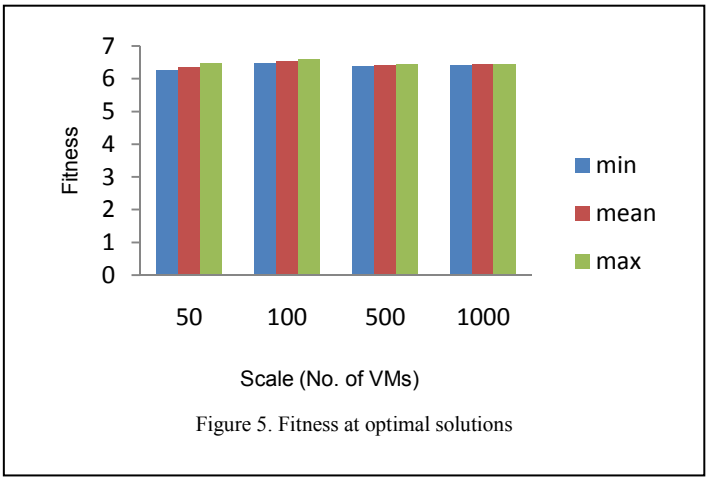


Figure 5. Fitness at optimal solutions