Dynamic Virtual Machine Allocation Based on Adaptive Genetic Algorithm

Oleksandr Rolik, Eduard Zharikov
Department of Automation and Control in Technical Systems, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
Kyiv, Ukraine
email:o.rolik@kpi.ua, email:zharikov.eduard@acts.kpi.ua

Sergii Telenyk, Volodymyr Samotyy
Department of Automatic Control and Information Technology, Faculty of Electrical and Computer Engineering, Cracow University of Technology
Cracow, Poland
email:stelenyk@pk.edu.pl, email:vsamotyy@pk.edu.pl

***Abstract* – The widespread use of the virtualization paradigm in modern data centers has increased the necessity of improving the management efficiency of virtual machine allocation on physical machines (PM). Modern service providers offer a large number of virtual machine types and settings. The density of virtual machine placement per physical server also complicates the solution of this problem. Under these conditions, for solving such kind of problems, the adaptive genetic algorithm (AGA) is proposed. The proposed algorithm uses parametric and algorithmic adaptation simultaneously by selecting the values for a genetic operator's parameters and by selecting the probabilities of applying these operators. The AGA is evaluated for the solution of virtual machine allocation problem and demonstrates efficiency compared to the classical and the controlled versions of genetic algorithm.**

Keywords – data center; genetic algorithm; virtual machine; resource management

I. INTRODUCTION

Data center resource management is an important and urgent problem at the present time. The growth in the number and complexity of modern data centers leads to an increasing number of virtual machines (VMs) and opens new challenges for management process automation. The Infrastructure as a Service (IaaS) paradigm enables customers to dynamically request the needed number of virtual machines based on their business requirements. One of the main tasks of managing resources in IaaS is the VMs allocation on the physical servers of the infrastructure. The VMs allocation process must be performed in a way that results in a reduction in the number of physical servers and a decrease in the energy consumption.

The use of genetic algorithms (GA) and their benefits compared with heuristic methods to solve data center resource management problems is shown in [1]. Classical genetic algorithms (CGA) have their own specifics as GA simultaneously use several types of genetic operators: unary, binary, and multiple. It is difficult to choose the strategy to generate values of probabilities for the use of certain operators so that their application gives positive results for the entire period of the GA. In addition, each operator has a set of parameters that influence the results of the algorithm, and to find the optimum values of these parameters is a rather difficult task.

In [1], a managed genetic algorithm (MGA) is proposed. It allows the adjustment of the parameters of the algorithm at all stages of the problem solving. In addition, the MGA does not suffer from the problems of the classic GA, such as degeneration of the population, getting into local extremes, etc. The main disadvantages of MGA are the need for the participation of an administrator and its application to a narrow class of problems. The solution to these problems of GA in general is not possible, therefore it is necessary to develop an effective strategy for the selection of the operator's parameters and for determining the probability of applying these operators for the entire period of the GA.

The remainder of the paper is organized as follows: in Section II, the related work is discussed, in Section III the problem of genetic algorithm adaptation is analyzed and an AGA is presented, in Section IV a particular case of the VMs allocation in the data center with a homogeneous configuration of the PMs is considered. This problem is proposed to be solved using an AGA. In this section, the results obtained by classical genetic algorithm, modified genetic algorithm and adaptive genetic algorithm for solving the considered problem are compared. Section V concludes the paper discussing the results and future research directions.

II. RELATED WORK

As stated in [2][3][4], there is a significant effort of research in the data center resource management field including resource provisioning, resource allocation, resource brokering, resource scheduling, resource mapping, and resource capacity planning. There are a lot of cloud computing frameworks and systems proposed that have specific mechanisms to provide and monitor resources, including those using heuristics and new methods such as load prediction mechanisms, considering imbalance of workload and virtual machine interference.

Genetic algorithms are widely used for solving computational problems of resource allocation in data centers, and produce sufficient productive solutions [5][6]. In these studies, one needs to determine the list of issues that

 ISBN: 978-1-61208-529-6

need to be figured out when using the GA to solve the problems of different dimensions and constraints.

During the last decade, many approaches to various VM placement problems have been proposed. In [7], the authors propose to model the server consolidation problem as a vector packing problem with conflicts using techniques inspired by grouping genetic algorithm. The algorithm has been tested in various scenarios and it allows to minimize the number of servers used for hosting applications within datacenters. Besides, it maximizes the packing efficiency of the servers utilized.

In [8], the authors consider the virtual machine packing problem as a multi-objective optimization problem and propose to solve it by using genetic algorithm as one of the meta-heuristics. The authors have implemented a virtual machine packing optimization mechanism based on genetic algorithm for a virtual cluster management system.

A hybrid genetic algorithm, using best fit decreasing strategy, was proposed in [9] to deal with infeasible solution due to the bin representation. The authors have proposed a new approach based on correcting infeasible chromosomes to prevent overflow of the bin. Hence, the new proposed chromosomes were suitable for the application of the genetic operators. They contributed to the execution time reduction. The proposed algorithm was evaluated on the VM placement problem and it lead to the usage of a minimum number of physical machines.

Energy consumption in the communication network is another subject of research related to the virtual machine placement problem. The approach based on genetic algorithm, that considers the energy consumption in both the servers and the communication network in the data center, was proposed in [10]. But the authors' assumption about network topology does not strictly correspond to the real data center networks.

To solve the VM placement problem in a cloud data center, the authors in [11] proposed to adopt a genetic algorithm using the future workloads prediction with Brown's quadratic exponential smoothing. But their online self-reconfiguration approach for reallocating VMs is focused on serving three types of Web transactions, namely browsing, shopping and ordering transactions.

To address fine-grained virtual machine resource allocation and reallocation problem, a two-level control system has been proposed in [12] to manage the mappings of workloads to VMs and the mappings of VMs to physical resources. An improved genetic algorithm with fuzzy multi-objective evaluation has been proposed to efficiently solve the VM placement problem, which is formulated as a multi-objective optimization problem of simultaneously minimizing total resource wastage, power consumption and thermal dissipation costs.

In [13] [14], the idea of genetic operator adaptation and adjustment of the probabilities of their use was proposed. The disadvantage of these approaches is that the adaptation of only one crossover operator was proposed. Another disadvantage is that the possible approaches to adaptation are used independently of each other.

As a rule, for the adaptation of the GA for specific tasks, a certain control parameter is used [15][16]. The effectiveness of the deterministic control has been proven for some tasks [13], but its generalization is problematic. Adaptive control [17] uses feedback to determine how the parameters should be changed. With adaptive control [14], it is necessary to introduce additional information that allows to adjust the behavior of the operators.

Today, the main efforts of GA researchers are focused on the adaptation of only one parameter. The most complete combination of management forms [16] was presented in [18], where the adaptation was performed simultaneously on three parameters: probability of mutation, crossover, and population size. Other forms of adaptive algorithms are presented in [19][20][21].

III. THE ADAPTIVE GENETIC ALGORITHM

The main feature of the proposed algorithm is the use of two adaptation strategies, so the process of obtaining the solution is a cyclical repetition of two stages. In the first stage, using the parametric adaptation, the algorithm selects the most productive settings for each of the operators. In the second stage, the parameter of performance is estimated and the algorithm selects the most efficient type of operator taking into account the results of algorithmic adaptation. If the resulting solution does not satisfy the specified criteria, the process is repeated from the first step.

The general formulation of the problem of genetic algorithm adaptation can be reduced to minimizing a function F that serves as a criterion for GA adaptation and depends on parameters such as: types of operators to be used in the evolutionary process, the frequency of use of these operators and the values of the parameters with which the operators are applied. Let us define the adaptation function as $F(M, C, \alpha_M, \alpha_C, \beta_M, \beta_C)$, where M, C are the parameters that govern the use of mutation and crossover operators, and take values from the set {0, 1}, and the equality of the parameter to 0 means that the operator does not take part in the evolutionary process, and the equality of the parameter to 1 means that the operator takes part in the evolution process; α_M, α_C are frequencies of the use of operators M and C respectively which take values from the interval [0; 1]; β_M, β_C are the sets of possible values of mutation and crossover operators.

Finding an explicit form of the function F, even for simple cases, requires a huge amount of computation that would negate all the benefits of GA. The use of structural adaptation within the solution of IT infrastructure management tasks is impossible, so when carrying out adaptation it is proposed to use some information regarding the properties of the function F. In the evolutionary process, such information will be the types of operators and parameters of these operators.

Genetic algorithms use several types of operators such as unary (mutation), binary (crossover), and multiple (multi-point crossover). Each type of operator has a set of parameters that affect its behavior, and GA begins with specific parameter values.

 ISBN: 978-1-61208-529-6

Suppose the set of types of operators is given as $D=\{d^1,...,d^z,...,d^Z\}$, where $z=\overline{1,Z}$ is the number of types of operators used in the GA. One such example is the modified crossover operator or the modified mutation operator. Each type of operator d^z, $z=\overline{1,Z}$ has a set of parameters $X(d^z)=(x_{z,1},...,x_{z,k},...,x_{z,Y})$, $k=\overline{1,Y}$. For example, the modified mutation operator has two parameters, which correspond to probabilities p_{10} and p_{01}, and crossover has only one parameter that is the number of crossover points.

To generate new generations, the GA uses genetic operators with parameters that correspond to its type. The parameters take their values from some set or interval. For modified mutation operator, the parameter's values will be selected from the interval [0, 1], whereas the parameter of crossover operator can take any positive integer value, less than $(n-1)$.

The parametric adaptation applied to increase the chances of survival and reproduction of the operators with the parameter values, showed the best results. Consider the AGA, which uses the types of operators from the set *D*. For each type of operator $d^z \in D$, $z=\overline{1,Z}$ the population of parameter values is constructed. For example, if modified mutation operator and modified crossover operator are defined for AGA, then $(alter\_rate, p_{10}, p_{01}) \in \{(0.1; 0.1; 0.9), (0.7; 0.8; 0), (0.4; 0.1; 0.1)\}$ is a fixed set of operator instances for mutations and $(c) \in \{(1),(2),(n-1)\}$ is fixed set of operator instances for crossover.

Let us denote a set of operators used by AGA as $O=\{op_1,...,op_h,...,op_H\}$, $h=\overline{1,H}$, where *H* is a number of operators in AGA.

The behavior of each operator is adjusted by changing its parameters using an evolutionary procedure for each operator. The evolutionary procedure operates in the space of possible values of the operator. After starting the AGA, changes in parameter values are made for each operator. Since relatively small search space is optimized, a GA is also used for the search of a set of the operator's values, leading to an improvement of the AGA results. GA is used for each individual operator. Let us denote as operator's stage of GA (OGA) the GA that implements searching of the best operator's values.

The population for each operator will consist of chromosomes, which are a set of possible values of the operator's parameters. A single-point crossover is used to generate new populations. The OGA that is used to search the best values in the space of a single type of the operator is running as the GA procedure and is referred to as the main stage of GA (MSGA). The MSGA is working on the direct solution search. The resulting solution obtained by the use of OGA is the initial data for MSGA. In this case, AGA is viewed as a set of the OGA and the MSGA cycles.

The algorithmic adaptation task is solved at the MSGA stage. This task is to increase the probability of use of operators that provide the best solutions. To evaluate the work of the operators, we introduce the following concepts.

The *event* – the use of a specific genetic operator. *Absolute improvement* – an event when the value of the objective function of the new generation is greater than the value of the objective function of previous generations. *Improvement* – the new generation has a better objective function value than the parent's one. *Stabilization of decision* – the value of the objective function of the new generation is slightly different from the parent's one for a number of epoch. *Degeneration* – the next generation has a worse value of the objective function than the parent's one, and none of the generations are better than the parental.

Let us introduce the parameters of performance, showing the efficiency of the operator in the current generation, and being used as a feedback for the evolutionary process. Based on the values of the performance indicators, the AGA corrects the values of probabilities of using the operators. The parameter of performance for the operator op_h, $h=\overline{1,H}$ is defined as a function of four variables $\pi_{op_h}(ae,e,pw,w)$, where *ae* is the number of absolute improvements, *e* is the number of improvements, *pw* is the amount of stabilized solutions, *w* is the amount of degeneration. The total number of events on the step of the AGA is defined as $N=ae+e+pw+w$.

The performance parameter is introduced in order to determine which of the operators provides the best solution in the current step of AGA. The frequency of using operators is taken into account with the relative frequencies of occurrence of a certain type of events. When $N>0$ the relative frequency is calculated as follows: for an absolute improvement – $\varphi_{ae}=ae/N$; improvement – $\varphi_e=e/N$; stabilization of decision – $\varphi_{pw}=pw/N$; degeneration – $\varphi_w=w/N$.

In this paper, we adopted the following procedure for comparing the performance parameters, which will be formulated as follows based on the example of the two operators. An operator will be more productive if it has a performance setting characterized by a large number of absolute improvements. If the number of absolute improvements is the same, the comparison is made on the number of improvements. If the number of improvements is also the same, the comparison is made on flat events. If this comparison does not allow to select the best operator, then the more productive will be the operator with less amount of degeneration. If there is an equal number of degenerations, then the mutation is used.

To determine the order of use of the operators we introduced the concept of reward. For each operator $op_h \in O$, $h=\overline{1,H}$ the award $\rho(op_h)$ is assigned, which increases as a result of the accumulation of positive experience. Primarily the operator with the greatest reward is used. The value $\rho(op_h)$, $h=\overline{1,H}$ is constantly updated, and the experience gained during recent tests is seen as more urgent.

 ISBN: 978-1-61208-529-6

Let $\chi(op_h),\ h=\overline{1,H}$ be the rank of operator op_h assigned depending on the performance such that the most productive type of operator gets the highest rank. Moreover, the assignment of the rank is made after completion of the OGA. The awards are updated at the end of each AGA epoch in accordance with formula (1)

$$\rho(op_h)=\delta\rho(op_h)+\beta+\gamma\chi(op_h), h=\overline{1,H}, \quad (1)$$

where δ is the attenuation factor that is constant in the set {0, 1}. And $\delta=0$ for the operator, that just came into the work, and $\delta=1$ means that all previous experience of the operator is fully taken into account. The amplification factor γ is to recognize the best operators. The coefficient β having a value less than γ, is commonly used to ensure that the operator is required to be used in AGA step.

An example for the two operators: crossover and mutation, as well as limitations on the duration of the periods as the amount of epoch is shown below.

Step 1: The setting of the stop condition of the algorithm, which may be the number of epochs of genetic algorithm or the algorithm duration.

Step 2. The initialization of the initial population randomly within the constraints (3)–(5) and the following rule: when accessing the population, each time its individuals are sorted in descending order of objective function value. The best representative of the population is saved as a stored solution. The best population is remembered as a stored population. Initially, the primary population is the stored population.

Step 3: The initialization of the population of the operator's values randomly subject to the restrictions imposed for each type of operator. For example, for the mutation parameters and the crossover the following values may be used $(alter\_rate, p_{10}, p_{01}) \in \{(0.1; 0.1; 0.1), (0.2; 0.2; 0.2), (0.3; 0.3; 0.3)\}$ and $(c) \in \{(1), (2), (3)\}$ respectively.

Step 4. The use of each value of the respective types of operators to generate intermediate populations, with the help of the objective function to select values which allow to achieve the greatest performance indicators for each of the type of operators. For example, for mutation it may be $(alter\_rate, p_{10}, p_{01}) = (0.1; 0.1; 0.1)$, for crossover it may be $(c) = (1)$.

In the case of improvement of the obtained results, the stored population is changed using the results obtained by means of the most productive of the two operators, and the stored solution is overwritten. If the performance of two operators is identical, the mutation operator has to be chosen. If this does not improve the initial solution, proceed to Step 3 and use OGA to adjust the values of the parameters using the crossover operator.

If it was not possible to improve the parameters of the operators by using crossover, then apply mutation to avoid possible falling into local extremes. The use of mutation parameter to adjust the values of parameters of the operators is made on a cycle by using the following rules. The rule for crossover is to increment the number of crossover points, and if the number of points is equal to $(n-1)$, to take the next number of crossover points equal to 1. The rule for mutation is to increase each of the values of mutation parameters by 0.1, and if the value of any of the parameters is 0.9, the next value is set to 0.1. If the decision is not improving, the algorithm finishes and the resulting solution is taken as the most productive.

Step 5: The ranks assigning to operators, calculation of awards, the use of the operators *M* times (the total number of MSGA steps is equal to 2×*M* in the first epoch, and 3×*M* in all subsequent) in descending order of reward value. If the use of the operator has improved the decision, the solution must be used as a new stored solution to the problem, and then continue with the improved population.

Step 6. After a predetermined number of AGA steps, the value of performance options is updated and the most productive operator is chosen. For example, if the operator $(alter\_rate, p_{10}, p_{01}) = (0.1; 0.1; 0.1)$ has been proven to be the most productive, it needs to be saved and used in the next epochs. Overwrite the stored solution.

Step 7. If the conditions of the AGA stop are not met, then go to step 3 and start a new epoch. On the stage of the OGA work, the most productive values for all types of operators must be chosen. For example, if $(alter\_rate, p_{10}, p_{01}) = (0.2; 0.2; 0.2)$ and $(c) = (1)$ for mutation and crossover respectively, then in MSGA step the work on population will be carried out for the three operators (the mutation operator $(alter\_rate, p_{10}, p_{01}) = (0.1; 0.1; 0.1)$ passes from the previous epoch). At the end of MSGA, count and compare the performance parameters, and choose the best of the three operators and go to the next epoch (Step 3).

Step 8. If the conditions of the AGA stop are fulfilled, then finish the job, use the current stored solution as a solution to the problem, otherwise go to step 3 and continue to work to complete the stop conditions.

In this paper, we used the combined stopping condition of the algorithm, which includes a predetermined time and the number of evolution periods in which the optimization result is not improving, and the growth of the objective function stops. GA aims to improve the outcome during the allotted time.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

A. Formulation of the problem

The authors consider a particular case (policy) of the VMs allocation in the data center with a homogeneous configuration of the PMs and propose to solve this problem using the adaptive genetic algorithm presented above. The VM allocation on the physical servers of the IT infrastructure relates to the problem of consolidation of computing resources. The case of using homogeneous PM configurations is chosen because it can be implemented within a single cluster, which may be considered as a unit of control in a data center. The mathematical model of the VMs allocation on the PMs is represented as follows.

 ISBN: 978-1-61208-529-6

The data center contains a set of PMs $N=\{N_1,\ldots,N_n\}$, where n is the number of PMs. $K=\{K_1,\ldots,K_m\}$ is a set of VMs that should be allocated to the PMs, where m is the number of VMs. Each PM N_i, $i=\overline{1,n}$, is characterized by two parameters that determine its computing capacity: Ω_i is the CPU capacity of the PM N_i, and Γ_i is the RAM capacity of the PM N_i. Each VM K_j, $j=\overline{1,m}$, has the computational resource requirements: ω_j is the CPU time, and γ_j is the RAM size. It is necessary also to determine the VM allocation matrix, $R=\|r_{ji}\|$, with the size of $m\times n$, where

$$r_{ji}=\begin{cases}1, \text{if VM } K_j \text{ is allocated on PM } N_i,\\ 0, \text{otherwise.}\end{cases} \quad (2)$$

The matrix R is a solution to the problem and determines the allocation of K VMs on the set N of PMs. The authors consider that all PMs in set N have identical specifications and, consequently, the same computing resources, so they assume that $\Omega_i = 1$ and $\Gamma_i = 1$ for all $i=\overline{1,n}$, that is

$$\{\Omega_i, G_i\}|_{N_i}=\{1,1\}, \text{ for all } i=\overline{1,n}. \quad (3)$$

This assumption allows the authors to make a transition from the measurement of PM computing resources in absolute values when the memory is measured in gigabytes and CPU frequency in GHz to a relative value. Then, the VM needs are defined as part of the PM's resources, recalculated in relation to the maximum possible value of 1. The number of resources allocated to a virtual machine is determined by the application requirements. The necessary resources for the VM are recalculated with respect to the physical server and they are part of it.

The authors consider also that resource needs of each VM do not exceed the capabilities of the PM

$$\omega_j \le 1 \text{ and } \gamma_j \le 1, \text{for each } j=\overline{1,m}. \quad (4)$$

When solving the problem of VMs allocation for all PMs from N, the following resource constraint must be satisfied

$$\sum_{j=1}^{m} r_{ji}\omega_j \le 1 \text{ and } \sum_{j=1}^{m} r_{ji}\gamma_j \le 1, \text{ for } i=\overline{1,n}. \quad (5)$$

Further, the authors introduce the vector $\vec{y}=\langle y_i\rangle$, $i=\overline{1,n}$, where

$$y_i=\begin{cases}1, \text{if at least one VM is allocated on } N_i,\\ 0, \text{otherwise.}\end{cases} \quad (6)$$

Then the optimum criterion for solving the problem of VM placement on PMs will be

$$\min \sum_{i=1}^{n} y_i, \quad (7)$$

that is the PMs should be filled with VMs so that the minimum number of PMs are involved.

When the criterion (7) is satisfied the total cost S of the data center and PMs maintenance and energy supply will be minimized.

The objective function can be represented as follows:

$$S=\sum_{i=1}^{n} s_i y_i, \quad (8)$$

where s_i is the maintenance and energy supply costs for the i-th PM.

In the case when the PMs in the data center have identical specifications (i.e., homogeneous), the expression (8) becomes

$$S=s\sum_{i=1}^{n} y_i, \quad (9)$$

where s is the maintenance and energy costs per PM.

Taking into account the previous description, the problem of K VMs allocation can be summarized as follows: it is necessary to place the VMs on data center PMs so that either the expression (8) or (9) reaches a minimum value.

The authors consider two cases, namely, the initial placement of the VMs and also their change in placement during the execution. The algorithm restarts when unused resources are detected in PM. If the number of unused resources on a PM is greater than the threshold, then that server is added to a consolidation list. The algorithm restarts if the total number of unused resources on the physical servers, included in the consolidation list, exceeds the resources of one PM. As a result, it is proposed to run the AGA not for all PMs, but for PMs in the consolidation list. All migrations initiated by the AGA at the previous stage must be completed. The selection of the threshold value is not considered in this work.

B. Evaluation

Experimental studies were performed on the data center resource allocation problem solution using three algorithms: the classical genetic algorithm, the managed genetic algorithm [1] and the adaptive genetic algorithm presented in this paper.

At the same time, studies were conducted for different ratios of the number of VMs and PMs resources. For this study, we considered three options for resource ratios that are close to reality:

- the case of disproportionate resource requirements when in relative units the requested amount of CPU time is much higher than the requested amount of

RAM, i.e., $\omega_j >> \gamma_j$, for all $j=\overline{1,m}$. This type of problem occurs when a large number of applications require complex calculations;

- the case of disproportionate resource requirements when requested units of CPU is much less than the requested amount of RAM, i.e., $\omega_j << \gamma_j$, for all $j=\overline{1,m}$. This problem occurs when the VMs with applications that require large amounts of data processing are located on the servers;
- the most common practical case is when the amount of the requested CPU and RAM for all VMs are distributed randomly in the range [0.05; 0.6].

The ranges of computer resources requested by VM for the different experiments are presented in Table 1.

TABLE I. THE RANGES OF COMPUTER RESOURCES REQUESTED BY VM

Experiment number	RAM limitation	CPU limitatiom
1	0.3—0.45	0.05—0.15
2	0.45—0.6	0.05—0.15
3	0.05—0.15	0.3—0.45
4	0.05—0.15	0.45—0.6
5	0.05—0.6	0.05—0.6

In [1], it has been proved that when the number of VMs is less than fifty the heuristic and genetic algorithms give approximately the same results, but with the increased number of VMs the genetic algorithm provides a better quality performance.

The evaluation of the quality of CGA, MGA and AGA algorithms is performed in terms of the number of the PMs released (turned off). As it is assumed that the physical servers are homogeneous, they consume the same amount of energy. It is assumed that to allocate each VM initially, a separate PM is deployed. Next, using the proposed algorithms placement of VMs on PMs are optimized with the assessment of the maximum number of released PMs for each of the algorithms. To compare the success of each algorithm according to the criterion (7), the authors assessed the number of unused physical servers, which were turned off. It is obvious that, as a result of each algorithm work for VM consolidation, the more PMs are turned off, the better.

Fig. 1 illustrates the dependence of the number of the PMs released on the problem dimension (number of VMs) in the case when the requested number of CPU and RAM for all VMs is randomly distributed in the range [0.05; 0.6]. The x-axis denotes the number of VMs, the y-axis denotes the number of PMs released.

For comparing the MGA and the AGA results, the concept of additional released PMs N_B is introduced. The value N_B is defined as the difference between the number of PMs N_{CGA}, released as a result of CGA, and the number of PMs N_{MGA} and N_{AGA} released using the MGA and the AGA respectively.

Thus, Fig. 2 shows a winning of the MGA and the AGA regarding the CGA as a function of a number of additional PMs released N_B from the dimension of the problem for different ratios of the resources requested.

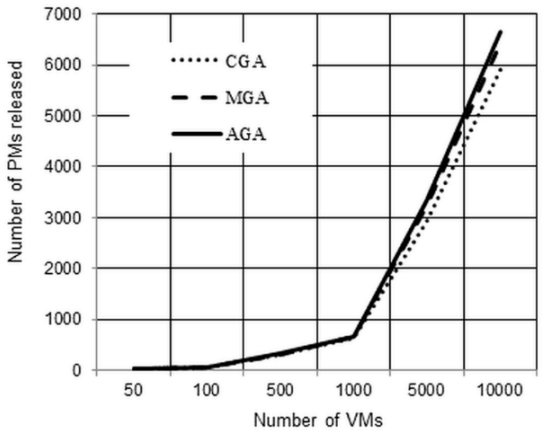

Figure 1. Dependence of the number of PMs released, from the problem dimension.

The x-axis represents the number of VMs that need to be placed on the PMs, y-axis represents the number of additional PMs released for each of the algorithms.

The data for the experiments were generated randomly with a uniform distribution law. The experimental results are shown in Fig. 2.

The analysis of the results shown in Fig. 2 leads to the following conclusions: (1) the use of the MGA and AGA is more effective than the use of the CGA; (2) the AGA always allows to get the best results on the VMs allocation, regardless of the experimental conditions; (3) in the case of dispersion over a wide range of requirements [0.05; 0.6] (Fig. 2 b), the use of the AGA is the most effective.

V. CONCLUSION AND FUTURE WORK

One of the most important problems in the modern virtualized environment is an allocation of virtual machines on physical servers. Taking into account the large number of types and virtual machine settings that modern service providers offer, as well as the high density of virtual machines per physical server the solution for this problem is complicated.

In this work, the virtual machine allocation problem is solved by using the adaptive genetic algorithm. The proposed adaptive genetic algorithm uses parametric and algorithmic adaptation simultaneously by selecting the values for genetic operator's parameters and by selecting the probabilities of applying these operators. The authors' added contribution regarding existing genetic algorithms is to apply the adaptation, which consists in the change of probability of using GA operators and in the change in the parameter values of these operators depending on the nature of the problems and on the results obtained during the problem solving.

The adaptive genetic algorithm is evaluated for virtual machine allocation problem solution and demonstrated efficiency compared to classical and controlled versions of the genetic algorithm. It is shown that the proposed algorithm allows for an equal number of epochs to get better results.

 ISBN: 978-1-61208-529-6

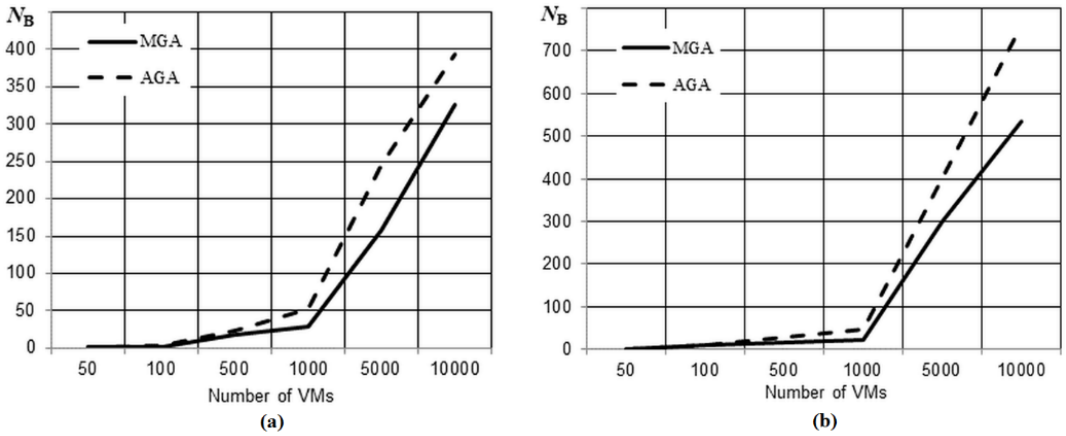

Figure 2. The dependence of the number of additionally released PMs on the dimension of the problem, when the requirements for the requested resources are in the ranges: (a) to the CPU – [0.05; 0.15], to the RAM – [0.3; 0.45]; (b) to the CPU – [0.05; 0.6], to the RAM – [0.05; 0.6]

For future work, the authors plan to develop the provisions of the adaptive genetic algorithm, to make recommendations on configuring the algorithm's parameters for specific kind of tasks, and apply an adaptive genetic algorithm for solving other tasks of the data center management.

REFERENCES

[1] S. Telenyk, O. Rolik, P. Savchenko, and M. Bodaniuk, "Managed genetic algorithm in problems of virtual machines allocation in the data center," Scientific Journal of ChSTU, No 2, pp. 104-113, 2011

[2] A. Hameed et al., "A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems," Computing, pp. 1–24, 2014.

[3] S. H. H. Madni, M. S. A. Latiff, Y. Coulibaly, and S. M. Abdulhamid, "Resource scheduling for infrastructure as a service (IaaS) in cloud computing: Challenges and opportunities," Journal of Network and Computer Applications, vol. 68, pp. 173–200, 2016.

[4] S. Singh and I. Chana, "A Survey on Resource Scheduling in Cloud Computing: Issues and Challenges," Journal of Grid Computing, pp. 1–48, 2016.

[5] S. Telenyk, O. Rolik, M. Bukasov, S. Androsov, and R. Rymar, "Control of Data Centers' Load and Resources Virtual Hosting," Scientific Journal of the Ternopil State Technical University, Vol 14, No 4, pp. 198–210, 2009.

[6] S. F. Telenik, A. I. Rolik, M. M. Bukasov, and S. A. Androsov, "Genetic algorithms of decision of tasks of management resources and loading of centers of processing of data," Automatic. Automation. Electrical engineering complexes and systems, No 1 (25), pp. 106–120, 2010.

[7] S. Agrawal, S. K. Bose, and S. Sundarrajan, "Grouping genetic algorithm for solving the server consolidation problem with conflicts," In GEC '09: Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation, New York, NY, USA, pp. 1-8, 2009.

[8] H. Nakada, T. Hirofuchi, H. Ogawa, and S. Itoh, "Toward virtual machine packing optimization based on genetic algorithm," In IWANN '09: Proceedings of the 10th International Work-Conference on Artificial Neural Networks, Berlin, Heidelberg, pp. 651-654, 2009.

[9] M. A. Kaaouache and S. Bouamama, "Solving bin Packing Problem with a Hybrid Genetic Algorithm for VM Placement in Cloud," Procedia Computer Science, Volume 60, pp. 1061-1069, 2015.

[10] G. Wu, M. Tang, Y. Tian, and W. Li, "Energy-Efficient Virtual Machine Placement in Data Centers by Genetic Algorithm," Neural Information Processing, Volume 7665 of the series Lecture Notes in Computer Science, Springer, pp. 315-323, 2012.

[11] H. Mi et al., "Online self-reconfiguration with performance guarantee for energy-efficient large-scale cloud computing data centers," Proc. of the IEEE International Conference on Services Computing, pp. 514–521, 2010.

[12] J. Xu and J. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," Proc. of the IEEE/ACM International Conference on Green Computing and Communications & 2010 IEEE/ACM International Conference on Cyber, Physical and Social Computing, pp. 179–188, 2010.

[13] T. Back, "Optimal Mutation Rates in Genetic Search," Fifth International Conference on Genetic Algorithms: University of Illinois at Urbana-Champaign, July 17–21, pp. 2–8, 1993.

[14] W. Spears, "Adapting Crossover in Evolutionary Algorithms," Proc. Of the 4th Annual Conference on Evolutionary Programming: San Diego, California, March 1–3, pp. 367–384, 1995.

[15] Z. Michalewich, "Genetic Algorithms + Data Structures = Evolution Programs," Berlin: Springer, 1996.

[16] Z. Michalewich and D. Fogel "How to Solve It: Modern Heuristics," Berlin: Springer, 2002.

[17] B. Julstrom, "What Have You Done for me Lately? Adapting Operator Probabilities in a Steady-State Genetic Algorithm," Proc. of the Sixth International Conference on Genetic Algorithms: University of Pittsburgh, July 15–19, pp. 81–87, 1995.

[18] J. Lis and M. Lis, "Self-adapting Parallel Genetic Algorithms with the Dynamic Mutation Probability, Crossover Rate and Population Size," Proc. of the 1st Polish National Conference on Evolutionary Computation. Oficina Wydawnica Politechniki, Warszawskiej, pp. 324–329, 1996.

[19] A. Kosorukoff, "Using incremental evaluation and adaptive choice of operators in a genetic algorithm," Proc. of the Genetic and Evolutionary Computation Conference: (GECCO-2002), New York, USA, July 9–13, p. 688, 2002.

[20] M. Pelikan, D. Goldberg, and S. Tsutsui, "Combining the strengths of Bayesian optimization algorithm and adaptive evolution strategies," Genetic and Evolutionary Computation Conference: (GECCO-2002), New York, USA, July 9–13, pp. 512–519, 2002.

[21] D. Thierens, "Adaptive mutation rate control schemes in genetic algorithms," Congress on Evolutionary Computation: CEC'02, Honolulu, Hawaii, May 12–17, pp. 980–985, 2002.

 ISBN: 978-1-61208-529-6