International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 12, December-2014

A Genetic Algorithm for Resource Provisioning of Virtual Service Based on Homogeneous Shared Hosting Platforms

Pham Nguyen Minh Nhut[1] ,
[1] Department of E-commerce,
Vietnam Korea Friendship Information Technology College,
Danang, Vietnam.

Le Van Son[2]
[2] Departement of Information,
Danang University of Education,
Danang University,
Danang, Vietnam.

***Abstract*- Optimal resource provisioning for virtual services in the Cloud computing is one of the most concerns nowadays. Multi-dimensional resource provisioning on a homogeneous shared hosting platform for virtual services is known as a NP-hard problem. Therefore, it is necessary to apply the metaheuristic algorithms for estimating the outcome of the problem. In this paper, we have effectively applied a genetic algorithm to solve the problem. We defined a fitness function with the goal of minimizing the number of physical machines, and compared our algorithm to standard algorithms of vector packing problem via emulation-based program in various scenarios. The experimental results show that: In the cases of large number of services, execution times of GA are shorter than the execution times of standard algorithms of vector packing problem.**

***Keywords*- Resource provisioning; Cloud computing; Genetic Algorithm (GA); Vector Packing; Virtual machine.**

I. INTRODUCTION

Virtual technology allows partitioning the resource of Y ($Y \geq 1$) physical machines into S ($S \geq 1$) virtual machines to execute the applications on demands. A system which consists of multiple physical machines with the same configuration connecting together for sharing resources is called a homogeneous shared hosting platform [3, 7, 9]. One of the challenges of this system facing is to minimize resources of the platform for virtual services while still ensuring the quality of service (QoS).

Resource management for shared hosting platforms has been investigated in many other studies [3, 4, 7, 9]. In particular, Urgaonkar *et al.* [3] propose a profiling technique for statistic of resource usage and minimum resource needs. Aron [4] and Casanova *et al.* [7] formulate the resource provisioning problem as a constrained optimization problem in which machines are considered as a monolithic resource. Stillwell *et al.* [9] further consider resource provisioning in a multi-dimention resource, however they focus only on efficiency of resource provisioning. They formulate the problem of resource allocation as a mixed integer linear program (MILP), where the objective is to maximize performance and fairness through a metric known as "minimum yield".

In this paper, we consider many aspects of resources and apply a linear objective function to minimize the number of physical machines. The resource provisioning problem is generally considered in both cases: static and dynamic, but we focus on solving the the problem for only static case (i.e., fixed resource needs). Moreover, the resource provisioning is known as a NP-hard problem, therefore this paper employs metaheuristic algorithms to solve it [2,8]. The key contributions of the paper are as follows:

1) Modeling a resource provisioning as a linear programming problem and computing the complexity of the problem.

2) Solving the propblem by applying GA and defining the fitness function in order to minimize the number of physical machines.

3) Evaluating and comparing experimental results with other results obtained by applying standard algorithms of vector packing problem [5, 6].

The rest of the paper is organized as follows: Section II presents a mathematical model of the problem as a linear programming problem and establishs the complexity of the problem. Section III solves the problem by applying the standard algorithms of vector packing problem. Our solution presents in Section IV by employing GA. Section V follows by experimental results and comparisons in various scenarios. Finally, Section VI concludes the paper and opens some future work.

II. RESOURCE PROVISIONING FOR VIRTUAL SERVICES

A. Resource and resource needs

Let's consider a homogeneous shared hosting platform in which a cluster of servers having the same configuration and being interconnected by a high-speed network devices is deployed for sharing resource to virtual services [nên tách thành 2 câu]. Each service [in the platform] operates as a virtual machine and the system ensures that service requests are dispatched to appropriate servers.

When users request virtual clusters (VC), the system responds by sets of virtual machine (VM). These VM instances run on physical machines (PMs) under the control of a hypersivor [1] and consume resources at different portions. The hypersivor can enforce specific resource consumption portions for different VMs running on the physical machine. A Resource Provider (RP) is responsible for making decisions whether to reject or admit a request, and allocates resource to each VM instance. Our goal is to design GA algorithm operated as a part of the RA to determine the minimum number of PMs based on resouce needs for virtual services.

To supply resource needs for virtual services, each PM provides several resources, i.e., CPU, RAM space, I/O bandwidth, disk space. In fact, each virtual sevice has two kinds of resource need: rigid and fluid [9]. A rigid need represents a specific fraction of required resource. The service cannot benefit from a larger fraction and cannot operate with a smaller fraction than a rigid need. A fluid need specifies the maximum fraction of a resource that the service could use if alone on the server. The service cannot benefit from a larger fraction, but can operate with a smaller fraction than a fluid need if the cost is reduced.

The ratio between the allocated resource and the fluid resource need is known as the yield of the fluid resource need, and we call their value simply the service yield. Within a service, the utilizations of all resources corresponding to fluid needs are linearly correlated [9]. Therefore, the service yield of each service is able to present by a value between 0 and 1. In particular, if the service yield is 0, the service will not be allocated any resource (due to the procedure of resource allocation). If the service yield is equal to 1, the service will be allocated resource as requied. However, it should be considered the lower bound on the yield of a service, which determines by QoS requirement(s). This constraint is defined by a service's fluid need multiplied by the service's QoS requirement(s), which is so-called a constrained fluid need. It is assumed that rigid resource needs are completely independent from fluid resource needs.

B. *Objective and constraints*

Assume that each service is represented by a single VM instance which has a fixed resource (static case). Multi-dimention resource provisioning problem (MRSP) is formulated as follows:

Let S_i be services, i = 1, 2, …, n; $S_i > 0$; Y_j be physical machines having the same configuration, j = 1, 2, …, m; $Y_j > 0$. Each physical machine provides D_k types of resources, $k = 1,..., d$. For each service i, r_{ik} denotes its resource need for resource type k, its value is between 0 and 1. We define a binary variable x_{ij} that is equal to 1 if service i runs on PM j and 0 otherwise. We use α_{ik} is a binary value that is equal 1 if r_{ik} is a rigid need, and 0 if r_{ik} is a fluid need; β_{ij} is the yield of service $_i$ on a physical machines $_j$; y_i is the number of PMs which is used for providing resource to service i. The resource provisioning problem represented by a linear programming problem with constraints and objective functions is as follows:

$$x_{ij} \in \{0,1\}, \beta_{ij} \in Q, \quad \forall i, j \qquad (1)$$

$$\sum_j x_{ij} = 1, \quad \forall i \qquad (2)$$

$$y_j \geq x_{ij}, \quad \forall i, j \qquad (3)$$

$$\sum_i (\beta_{ij}(1-\alpha_{ik}) + \alpha_{ik}) r_{ik} x_{ij} \leq 1, \forall k, j \qquad (4)$$

and, object function is $\min \sum_j y_j$ (5)

Constraint (1) defines the domain of the variables. Constraint (2) determines the state at which there exists a service i running on a physical machine j or not. Constraint (3) specifies the state at which a physical machine j is being used or not. Constraint (4) represents the state at which the fraction of total resource needs for service i is always less than or equal to the total resource of the physical machine j. The Eq. (4) implies that if resource need r_{ik} is fluid, then $\alpha_{ik} = 0$ and the fraction of resource D_k used on the physical machine j is $\beta_{ij} \times r_{ik}$. If resource need r_{ik} is rigid, then $\alpha_{ik} = 1$ and the fraction of resource D_k used on the physical machine j is r_{ik}.

Finally, constraint (5) is the objective function which denotes the number of physical machines used for providing resources to the virtual service. The objective is to minimize y_j.

C. *Computational Complexity*

To determine the computational complexity of MRSP problem, let's consider MRSP-Dec, the decision problem associated with MRSP can be stated as: *Is it possible to assign S_i services, each of which has a resource needs r_{ik} to Y_j physical machine*?

Theorem 1. *The decision problem MRSP-Dec is NP-C.*

Proof.

(i) It can be clearly seen that the problem MRSP-Dec is NP. Because the solution, if it exists, can be verified in polynomial time.

(ii) Consider the vector packing problem presented in [5, 6] as NP-C: Given a set ***A*** consisting of elements of the d-dimensional vector represented by a d-tuple: $(a_1^i, a_2^i, ..., a_d^i)$ and set ***B*** consisting of elements of the d-dimensional vector represented by d-tuple: (1,1, ..., 1), put the elements of the set ***A*** into the set ***B*** such that $\sum_{i \in B} a_j^i \leq 1, \forall j = 1,...,d$.

A MRSP problem is reduced to vector packing problem as follows: Let the number of physical machine be ***B*** (i.e., ***Y***=***B***), the service be ***A*** (i.e., ***S***=***A***), the number of resource types be j (i.e., k=j) and the resource need of service i of the resource k be a_j^i (i.e., r_{ik}= a_j^i). For each service i, set $\alpha_{ik} = 0$ (i.e., the fluid need is considered only,

for the rigid need, the proof will be similar). Clearly, the vector packing problem provides a solution to the MRSP-Dec problem. In contrast, a solution of the MRSP -Dec problem will provide a solution to the vector packing problem. From (i) and (ii), Theorem 1 has been proofed.

As proofed above, this problem is a combinatorial optimization problem which is known as NP-hard. To solve this kind of problem, several solutions have been proposed such as: The heuristic approaches applied for finding the best solution; The local searching employed to look for a local optimal solution; The approximation approaches by applying metahueristic algorithms. The next section presents the standard heuristic algorithms of the vector packing problem and a genetic algorithm for solving this problem.

III. SOLUTIONS BASED ON STANDARD ALGORITHMS OF VECTOR PACKING PROBLEM

The basic idea of the algorithms of vector packing problem applied to solve MRSP is that putting D_k-dimension S_i vectors into Y_J (in this case the Si is virtual service, Y_J is the number of physical machines and D_k is the types of resource). In this paper, we consider the standard algorithms: *First Fit*, *Best Fit* proposed in [5]. Before putting elements of S_i into Y_j, elements S_i are sorted in a descending order based on the following criterias:

- Lexicographical: Given $k \geq 1$, $\theta^k = \{ (S_1,S_2,\ldots,S_k), \forall i, 0 \leq S_i \leq 1\}$ and $a, b \in \theta^k$. $a \leq b$ iff a = b or the first nonzero component of b-a is positive
- Maximum Component: Given $k \geq 1$, $\theta^k = \{ (S_1,S_2,\ldots,S_k), \forall i, 0 \leq S_i \leq 1\}$ and $a, b \in \theta^k$. $a \leq b$ iff the maximum component in b is not less than the maximum component of a.
- Maximum Sum: Given $k \geq 1$, $\theta^k = \{ (S_1,S_2,\ldots,S_k), \forall i, 0 \leq S_i \leq 1\}$ and $a, b \in \theta^k$. $a \leq b$ iff the sum of components of b is not less than the sum of components of a.

Based on standard algorithms of vector packing problem, we construct algorithms to solve MRSP problem as follows:

Algorithm 1

Input: A set of services S_i (each service has resource need r_{ik} corresponding to the type of needs α ik) and the set of physical machines Y_j corresponding service yield β_{ij}.

Output: A set of minimum physical machines y_j (respectively, $x_{ij} = 1$).

The steps of the algorithm:

- Step 1: Based on the resource need r_{ik}, building D_k-dimensional vector S_i with r_{ik} elements.
- Step 2: Apply Criteria 1 to sort elements of vector S_i in a descending order.
- Step 3: Apply the *First Fit*, *Best Fit* algorithms to place the elements of the vector S_i into the physical machine Y_j, such that:

$$\sum_i (\beta_{ij}(1-\alpha_{ik})+\alpha_{ik}) r_{ik} x_{ij} \leq 1 \quad , \quad \forall k, j$$

- Step 4: If the resource needs have not completed then go to step 3.
- Step 5: If the resource needs have been completed then the output is a set of physical machines Y_j (this is the outcome of the objective function of the MRSP problem).

The combinatorial algorithms constructed from greedy algorithms, i.e., *First Fit*, *Best Fit* based on the 3 Criteria, therefore we have 6 algorithms including: *BestFitDesSum*, *BestFitDesMax*, *BestFitDesLex*, *FirstFitDesSum*, *FirstFitDesMax* và *FirstFitDesLex*. If D_k is considered as a constant, those algorithms have the computational complexity of O(S.log$Y$+$S.Y$).

IV. SOLUTION BASED ON GENETIC ALGORITHM

A. Introduction to GA

GA [2, 8, 10, 11] is based on the law of biological evolution of life populations. Individuals pass a process of development and reproduction to the creation of new individual for the next generation. In the process of evolution, bad individuals (based on a given criteria, so-called fitness) are eliminated. In contrast, qualified individuals are retained. Some concepts related to genetic algorithm are as follows:

- Individual representation: It is the representation of individuals so that each individual is a solution of the current problem.
- Fitness evaluation: It is the evaluation of the adaptability of each individual for a problem. The evaluation is based on the fitness function.
- Crossover operation: It is the process of creating new individual based on current individual (called father-mother individual). Two child individual are generated by swapping from the father-mother individual genes.
- Mutation: It is a process of creating new individual from a given individual by changing some of its genes.
- Selection and Replacement: It is the process of selecting individuals from the current populations to create the next generation. In this process, if the individual has a greater than or equal criteria adaptation, this individual will be retained. Adaptability of individuals in a population is more complete more.
- Terminate condition: A genetic algorithm is the random process, the algorithm can not be sure to stop after finite steps. Therefore, we typically have to define the terminate conditions for the algorithm.

B. GA for resource provisioning

As previously discussed, a genetic algorithm allows us to optimize the NP-hard problems in certain time. This section focuses on GA applied for solving the resource provisioning problem for the virtual service as described in Section II.

- Individual representation: Each individual is represented by a one-dimensional integer array S_i with length of n (S_i: virtual service; i = 1, ..., n), in which the i-th element equal to j if service S_i is allocated to the physical machine Y_j(Y_j: physical machine; j = 1, ..., m).
- Initial population generation: Generate P (P is a designed parameter) initial random individuals. An initial chromosome is obtained by randomly assigning a service S_i to a physical machine Y_j.
- Fitness function: Let $F_k(Y_j)$ be the total percentage of resource need D_k of the service S_i that a physical machine Y_j requires to supply for service S_i:

$$F_k(Y_j) = \left(\sum_{i \in Y_j} (\beta_{ij}(1-\alpha_{ik}) + \alpha_{ik}) r_{ik} x_{ij} \right)^K \quad (6)$$

where K (K> 1) is a exponental constant which expresses concentration on the physical machines compared to the less-filled physical machines. The larger the K is, the quicker the algorithm converges. a few well-filled physical machines are preferred to a collection about equally filled physical machines. But, leading to premature convergence ability of the algorithm. Experimetally, the optimal value of K is 2, i.e., K = 2. Therefore, the fitness function of the GA for the whole resource need D_k is identified as

$$f_{VSMSA} = \sum_{k=1}^{d} \left(\frac{1}{m} \sum_{j=1}^{m} F_k(Y_j) \right) \quad (7)$$

where m is the number of utilised physical machines.

- Crossover Operation: We use a one-point crossover operator by which two parent chromosomes are fragmented into two segments. Then two new chromosomes are generated by concatenating of chromosome segments fragmented from their parents. Crossover process is applied for all possible pairs of individuals with the probability of p_c (p_c is the designed parameter).
- Mutation Operation: The mutation operation randomly swaps between two services of two different physical machines. Mutation process is applied for every individual with the probability of p_m (p_m is a design parameter).
- Selection of individuals for the next generation: This is a selection P individuals for the next generation. A newly generated chromosome (after the processes of initial, mutation, and crossover) may not be ineligible, so-called ineligible chromosome. But, They are still in use for new population in the proposed algorithm. After generating an ineligible chromosome, a greedy algorithm is applied to make it becoming an eligible chromosome. This algorithm goes through the physical machines in an arbitrary order, for any overloaded physical machines, the algoritm attempts to move services to others which have fewer loaded services. This approach has been practically evaluated, by which it helps to reduce the diversity of the chromosome population, and trends to an eligible resource provisioning.
- Terminate condition: The algorithm will stop after G generation (G is a designed parameter) or the average value of the Fitness function of individuals reaches an converge to the expected value.

In summary, the genetic algorithm for resource provisioning consists of the following steps:

Algorithm 2

Input: A set of services S_i (each service has resource need r_{ik} corresponding to the type of needs α ik) and the set of physical machines Y_j corresponding service yield β_{ij}.

Output: A set of minimum physical machines y_j (respectively, x_{ij} = 1).

The steps of the algorithm:

```
1:  begin
2:    t ← 0;
3:    Initializing the first population P(t);
4:    Calculating the fitness function of each chromosome
in
       population P(t);
5:   until (the terminate conditions is not satisfied)
6:      t ← t+1;
7:         Selection of P(t) based on roulette mechanism
and
        fitness value at P(t-1);
8:         One-point crossover operation on P(t) to obtain
Q(t);
9:      Mutation operation in the P(t) to obtain R(t);
10:     Selection from P(t-1)∪Q(t)∪R(t) to obtain P(t);
11:   end until;
12: end
```

C. The computational complexity of the algorithm

The GA algorithm parameters are set before running. Let n be the number of services, m be the number of physical machines, and d be the dimensions of resource. The computational complexity of the algorithm is identified as

- Generating the first population: P.O ($n.m.d$).
- Calculating the fitness function: O ($n.m.d$).
- Crossover operation: P.(P-1).O($n.m.d$)=P^2.O($n.m.d$)

- Mutation: P.O($n.m.d$)
- Selection: O ($P.logP$)

Therefore, the computational complexity of the algorithm is

P.O($n.m.d$)+G.(P.(O($n.m.d$))+P^2.(O($n.m.d$+$P.logP$+O($n.m.d$))+P.O($n.m.d$)+O($P.logP$))=$G.P^2$.O($n.m.d$).

V. NUMNERICAL RESULTS AND EVALUATIONS

A. Simulation setup

To evaluate the proposed algorithm, a set of experimental random instances are generated based on [9] as follows: Given S services and Y physical machines with D resource dimensions. For each service, the numbers of rigid need are $D/2$ and the numbers of fluid need are $D/2$. All resource needs are sampled from a normal probability distribution with mean μ and standard deviation δ. Each service has a QoS request with the probability of ρ.

Assume that the value of the parameters are as follows: Service yield β_{ij} = 0.5, numbers of service S = 32, 64, 128, 256, 512, resource dimensions D = 6 (the numbers of rigid need = the numbers of fluid need =3), μ = 0.5, δ = 0.25, 0.5, 1.0, ρ = 0.25, 0.5, 1.0 and all QoS requirements to be 0.5 (i.e., half of the service's fluid needs must be met). Experiments results obtained by using other values of parameters are similar. This setup corresponds to 1×5×1×1×3×3= 45 scenarios. For each scenario, 100 random samples are generated resulting in a total of 4500 input individual instances used for evaluation.

Two metrics were employed for evaluation:

- The minimum number of physical machines that corresponding to the 5 values of services S = 32; 64; 128; 256; 512.
- The average execution times of the algorithm in seconds. The values of the two metrics are averaged from 900 (i.e., 3 × 3 × 100) experimental instances. The algorithms were coded in C++ language and ran on an Intel Core Duo 1.86 GHz and 2 GB RAM.

We set the size of population P = 100, G = 2000 generations, the probability of mutation p_m = 0.1, the probability of crossover p_c = 0.25 and constant K = 2. These parameters are estimated empirically based on experience.

C. Simulation results and evaluations

The minimum values of the physical machine (objective function values) and the standard algorithm execution times of vector packing problem are presented in Table I and Table II. The results obtained by using the GA are presented in Table III.

TABLE I. COMPARISON OF MINIMUM PHYSICAL MACHINES

Algorithms	Number of services				
	S=32	S=64	S=128	S=256	S=512
FirstFitDesMax	24	47	90	174	344
FirstFitDesLex	24	47	90	174	344
FirstFitDesSum	24	47	90	174	344
BestFitDesMax	24	47	90	174	344
BestFitDesLex	24	47	89	170	324
BestFitDesSum	24	47	90	174	344

TABLE II. COMPARISON OF EXECUTION TIMES (S)

Algorithms	Number of services				
	S=32	S=64	S=128	S=256	S=512
FirstFitDesMax	0.00009	0.00107	0.00303	0.01076	0.03193
FirstFitDesLex	0.00010	0.00114	0.00214	0.00827	0.02529
FirstFitDesSum	0.00016	0.00113	0.00268	0.00932	0.02791
BestFitDesMax	0.00121	0.00254	0.00833	0.03741	0.13107
BestFitDesLex	0.00123	0.00232	0.00783	0.03500	0.12253
BestFitDesSum	0.00128	0.00234	0.00800	0.03693	0.13380

TABLE III. EXECUTION TIMES AND MINIMUM PHYSICAL MACHINES OF GA

Number of services	S=32	S=64	S=128	S=256	S=512
Execution times (s)	0.0010	0.00307	0.02076	0.04126	0.08930
Minimum physical machines	24	47	90	174	344

From results shown in Table III and the results obtained from 6 standard algorithms of vector packing problem in Table I and Table II. It is clearly seen that with the same set of data, the execution time of the GA is longer than others. However, in the cases of large number of services (512 services), execution times of GA are shorter than *BestFitDesMax*, *BestFitDesLex*, *BestFitDesSum* algorithms. The objective function values of (number of physical machines) all algorithms are similar (except for *BestFixDesLex* algorithm with a better value). Moreover, the execution times of the GA are short and therefore it can be applied in practice.

VI. CONCLUSION

The paper has dicussed static resource provisioning based on a homogeneous shared hosting platform for virtual services with the optimal constraints and QoS requirements; each service is considered as a single virtual machine. On the basis of the optimal problem, we have proposed a GA to minimize the number of physical machines. The execution times of the GA are short and therefore it can be applied in practice. For the future work, the proposed model will be extended for heterogeneous platform.

REFERENCES

[1] L. V. Sơn, P. N. M. Nhut, "Some issuses of providing virtual machines resources base on infrastructure of cloud computing", *Journal of Science and Technology*, Danang University, 5(11), (2012), pp. 63-71. (chỉnh lại tất cả Ref. theo định dạng này !!)

[2] T.T.T Hang, L.T. Vinh, H.C. Thanh, N.T. Thuy, "Optimization for multi-objective scheduling in grid computing system", *Journal of Computer Science and Cybernetics*, Vietnam Academy of Science and Technology, 25 (1) (2009), pp. 79-87.

[3] B. Urgaonkar, P. Shenoy and T. Roscoe, "Resource Overbooking and Application Profiling in Shared Hosting Platforms", *SIGOPS Operating Systems Review*, 36(SI), (2002) pp.239-254.

[4] M. Aron, P. Druschel and W. Zwaenepoel, "Cluster Reserves: A Mechanism for Resource Management in Cluster-based Network Servers", *In Proceedings of the 2000 ACM Sigmetrics International Conference on Measurement and Modeling of Computer Systems*, New York, USA, (2000), pp.90-101.

[5] L. T. Kou, G. Markowsky, "Multidimensional Bin Packing Algorithms", *IBM Journal of Research and Development*, 21(5) (1977) pp.443 - 448.

[6] K. Maruyama, S. K. Chang and D. T. Tang, "A general packing algorithm for multidimensional resource requirements", *International Journal of Computer and Information Sciences*, 6(2) (1977) 131-149.

[7] Henri Casanova, David Schanzenbach, Mark Stillwell, Frédéric Vivien, "Resource provisioning using Virtual Clusters", Research Report No 2008-33, October 2008.

[8] Christian Blum, Andrea Roli, Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison, *ACM Computing Surveys*, 35(3), (2003), pp. 268-308.

[9] Mark Stillwell, David Schanzenbach, Frédéric Vivien, Henri Casanova, "Resource provisioning algorithms for virtualized service hosting platforms", *Journal Parallel Distrib. Comput*., 70 (2010) pp.962–974.

[10] Holland J., "Adaptation in natural and artificial systems", *University of Michigan press*, Ann Arbor, MI, MIT press, Cambridge, MA, (1975,1992).

[11] Melanie Mitchel, "An introduction to genetic algorithms", MIT Press Cambridge, MA, USA, 1996.