TECHNICAL TRANSACTIONS | CZASOPISMO TECHNICZNE
ELECTRICAL ENGINEERING | ELEKTROTECHNIKA

3-E/2016

DOI: 10.4467/2353737XCT.16.279.6078

SERGII TELENYK[*], MAKSYM BUKASOV[**]

DATA CENTER RESOURCE MANAGEMENT FOR SAAS

ZARZĄDZANIE ZASOBAMI CENTRUM DANYCH DLA SAAS

A b s t r a c t

The paper summarizes models and methods of data center resource management for SaaS. An approach for the allocation of computing resources for single-tenant SaaS was proposed. Different cases of this problem for excess and lack of computing resources were considered. Those problems belong to the classes of linear and nonlinear Boolean programming. In order to solve the mentioned problems, heuristic and genetic algorithms have been proposed. A comparison of their effectiveness was made.

Keywords: data center, SaaS, resource management, virtualization

S t r e s z c z e n i e

W artykule przedstawiono modele i metody zarządzania zasobami centrum danych dla SaaS. Zaproponowano podejście do alokacji zasobów obliczeniowych dla jednego dzierżawcy SaaS. Rozpatrzono różne przypadki tego problemu dla nadmiaru i braku zasobów obliczeniowych. Problemy te należą do klasy liniowego i nieliniowego programowania boolowskiego (logicznego). Do rozwiązania wymienionego problemu zaproponowano algorytmy heurystyczne i genetyczne. Porównano ich skuteczność.

Słowa kluczowe: centrum danych, SaaS, zarządzanie zasobami, wirtualizacja

[*] Prof. D.Sc. Ph.D. Sergii Telenyk, Department of Automatic Control and Information Technology, Faculty of Electrical and Computer Engineering, Cracow University of Technology.

[**] Ph.D. Maksym Bukasov, Departament of Automation and Control in Technical Systems, National Technical University of Ukraine “Kyiv Polytechnic Institute”.

Definitions

S_i – physical server in cluster, $i = 1, ..., n$,
e_i – power consumption of server S_i, $i = 1, ..., n$,
R_k – resource of type k (e.g. CPU, memory), $k = 1, ..., l$,
r_{ki} – amount of resource R_k on server S_i,
V_j – virtual machine, $j = 1, ..., m$,
w_j – importance weight of applications in V_j, $j = 1, ..., m$,
p_{kj} – demands of V_j in resources R_k,
x_{ij} – Boolean variable; $x_{ij} =1$ for V_j placed on S_i, otherwise $x_{ij} =0$.

1. Introduction

Software as a service (SaaS) is a software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted. Multi-tenant Software as a Service is an architecture where multiple customers (tenants) share the same application instances. Those instances are typically logically divided to prevent the customers from accessing each other's data. There are considerable savings in hardware and power consumption because of the use of shared resources. Financial efficiency is a major advantage of multi-tenant SaaS [1]. Single-tenant Software as a Service is an architecture where each customer has his own application instance (i.e. a separate physical or virtual machine for each customer is needed). By having a single hosted instance, the customer can tweak and customize the software to meet their needs. With single-tenant SaaS, there is only one instance of application for a customer and it is impossible to affect another customer's tasks, so reliability and security is a major advantage of single-tenant SaaS. Single-tenant systems are generally more expensive than multi-tenant solutions because they are not the most efficient use of resources, unless fully loaded. To solve these problems, it is necessary to create flexible solutions that are built on load balancing and resource allocation [2]. This in turn requires appropriate mathematical models and methods [3–6].

2. The problem

Consumers and hosting companies agree on service level requirements, which usually include: the availability and manageability of IT infrastructure, data integrity, security, reliability, scalability. Achievement of user requirements for the lowest cost is the essence of the problem of development and functioning of the IT infrastructure.

Methods of adding more resources for a particular application fall into two broad categories: horizontal and vertical scaling [7, 8]. To scale horizontally (or scale out/in) means to add more nodes to (or remove nodes from) a system, such as adding a new computer to a distributed software application. To scale vertically (scale up/down) means to add resources to (or remove resources from) a single node in a system, typically involving the addition of CPUs or memory to a single computer or a Virtual Machine (VM).

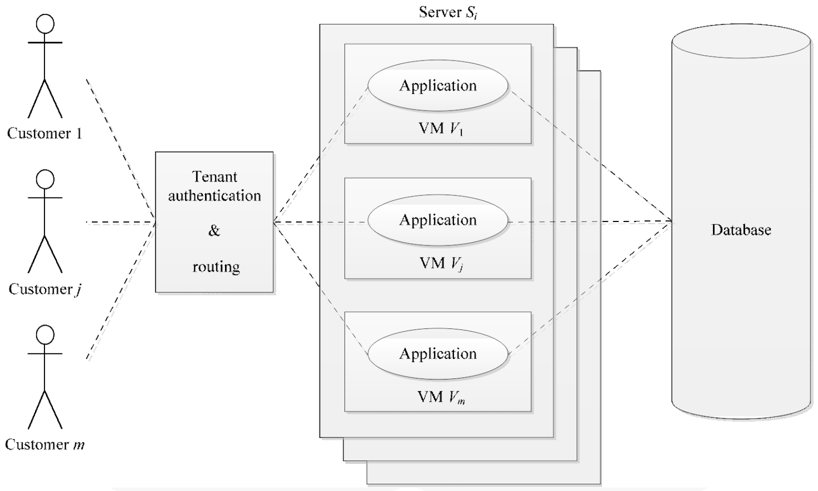

Fig. 1. Resource allocation for single-tenant SaaS

In order to meet the customer's requirements and to maximize own profit in the best way possible, the provider can solve one of the next three problems depending on resource availability.

Problem 1. If vertical scaling is not used too often, it makes sense to find a VM distribution where they will be allocated on the servers most tightly to free up most of the computing resources for other tasks or to provide a high level of energy efficiency by turning off unused servers. Also, this makes it easy to perform horizontal scaling by adding new VM with application instances to unused servers.

Firstly, demands in resources of all VMs on server S_i must not exceed the available resources of this server:

$$\sum_{j=1}^{m} x_{ij} p_{kj} \le r_{ki};\ k = 1, ..., l;\ i = 1, ..., n\,. \tag{1}$$

Secondly, all VMs should be placed on, and each of them should be placed no more than on one server:

$$\sum_{i=1}^{n} x_{ij} = 1;\ j = 1, ..., m\,. \tag{2}$$

Let us define the indication of no VMs on server S_i as:

$$\prod_{j=1}^{m} \overline{x_{ij}} = 1,\ i = 1, ..., n\,. \tag{3}$$

Then, in order to minimize power consumption of servers, let us use the following criteria:

$$\max \sum_{i=1}^{n} e_i \prod_{j=1}^{m} \overline{x_{ij}}\,, \tag{4}$$

and formulate the problem as follows: satisfy (4) under constraints (1), (2).

Problem 2. If application's load can often change, it is necessary to provide the possibility of the vertical scaling i.e. adding computing resources to VMs. In order to complete this task quickly (without migration of VMs to a different physical server), we need to assure that the server will already have free (reserved) resources. To do this, instead of constraints (1), we will use:

$$\sum_{j=1}^{m} x_{ij} p_{kj} \le r_{ki} - \Delta r_{ki};\ k = 1, ..., l;\ i = 1, ..., n\,, \tag{5}$$

where:

Δr_{ki} – reserved resources of type k on server S_i.

Then, let us formulate the problem as follows: satisfy (4) under constraints (2), (5).

Problem 3. If no acceptable solution to Problem 1 was found due to a lack of resources, we have a situation in which it is not possible to place all the VMs under resource constraints (1). In this case, it would be appropriate to allocate all the needed resources among the most important VMs at the first phase, and residual resources among the rest of VMs at the second phase. So instead of constraints (2), the following should be used:

$$\sum_{i=1}^{n} x_{ij} \le 1;\ j = 1,...,m\,, \tag{6}$$

and criteria:

$$\max \sum_{i=1}^{n} \sum_{j=1}^{m} x_{ij} w_j\ . \tag{7}$$

Then, let us formulate the problem as follows: satisfy (7) under constraints (1), (6).

3. Resources allocation methods

Problems described above belong to a widespread class of 0–1 programming problems. To solve these problems, we will use a heuristic greedy algorithm and guided genetic algorithm [9].

Heuristic algorithm. Since we are interested in the densest distribution of VM through the servers, let us formulate an idea of the algorithm as follows:

```
while (the list of unallocated VM is not empty)
{
        try to place VM with the highest requirements to CPU;
        try to place VM with the highest requirements to RAM;
}
```

Genetic algorithm GA. Since each *VM* can be placed not more than on one server, for encoding genes, let us move from $n*m$ matrix x_{ij} of Boolean variables to the length m vector y_j of discrete variables. Each element of that vector is the server's number $I = 1,...,n$, which contains the appropriate *VM*. For example:

$$x_{ij} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$y_j = [3\ 2\ 4\ 2\ 1\ 3\ 1\ 1].$$

This method of coding genes allows, firstly, to reduce the dimension of the problem, and secondly, to provide automatic execution of constraints (3). Herewith, the mutation operation will correspond to *VM* transferring from one server to another and the crossover operation – to multiple *VM* migrating between servers [10].

To provide the possibility of turning off unused servers in problems 1 and 2, we must provide the most dense allocation of VM on servers under resource constraints (1). Let us define the fitness function for this task as:

$$f = \sum_{i=1}^{n} F_i , \tag{8}$$

$$F_i = \begin{cases} A_i^2, A_i \geq 0 \\ -B(A_i), A_i < 0 \end{cases}, \tag{9}$$

where:

A_i – free resources on server S_i,
B – penalty function.

4. Experimental results

The effectiveness of the proposed algorithms was carried out as follows. Cluster of n servers (n = 4, 8, 12, 16) had been simulated. Each server in cluster had 8 CPU cores and 32 Gb of RAM. VMs with random demands to CPU and RAM had been created and placed on the first server where it can be placed according to resource constraints (3). If a new VM cannot be placed on either server, we had tried to optimize VMs allocation scheme. Two series of experiments, differed by the spread of demands to CPU and RAM, were done. For each series 100 samples were randomly generated. Each problem was solved with heuristic and genetic algorithm. Total numbers of servers that can be turned off after migration of VMs in all 100 experiments are shown in the Table 1 and Fig. 2.

Table 1

Experimental results

n	Serie 1 (100 experiments, low spread) CPUmin = 3, CPUmax = 5, RAMmin = 10, RAMmax = 20		Serie 2 (100 experiments, high spread) CPUmin = 1, CPUmax = 8, RAMmin = 1, RAMmax = 32	
	Heuristic algorithm	Genetic algorithm	Heuristic algorithm	Genetic algorithm
4	14	102	3	112
8	42	212	18	239
12	72	295	38	366
16	88	379	62	487

The number of servers in cluster deferred on *x*-axis. On *y*-axis – the total number of servers in 100 experiments that can be successfully turned off. The results of heuristic algorithm are labeled as «H», genetic – as «GA», series of experiments are labeled with numbers 1 and 2.

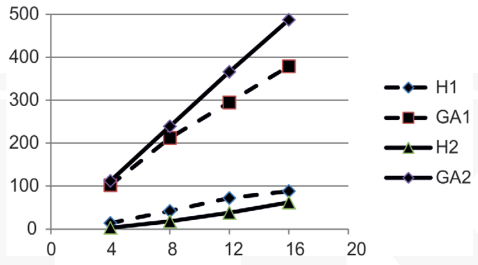

Fig. 2. Comparison of heuristic and genetic algorithms effectiveness

5. Conclusions

Models and methods for solving the resource allocation problem in data centers providing single-tenancy SaaS were proposed.

Formulated problems were reduced to problems of Boolean programming. Greedy heuristic and genetic algorithms were proposed. Genetic algorithm provides much better results for both series of experiments.

Results of the experiments confirmed the efficiency of the proposed approach for SaaS service providers.

References

[1] Lebrun C., *The Benefits of Multi-tenancy to Manage IT & Communication Expenses*, Cimpl Blog, 2016, http://blog.cimpl.com/the-benefits-of-multi-tenancy-to-manage-it-communication-expenses.

[2] Resource Management with VMware DRS, VMware Infrastructure, 2006, https://www.vmware.com/pdf/vmware_drs_wp.pdf.

[3] Katyal M., Mishra A., *A Comparative Study of Load Balancing Algorithms in Cloud Computing Environment*, International Journal of Distributed and Cloud Computing, Vol. 1, Issue 2, 2013.

[4] Beloglazov A., Buyya R., *Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers*, Concurrency and Computation: Practice and Experience, vol. 24, no. 13, 2012, 1397–1420.

[5] Horri A., Mozafari M.S., Dastghaibyfard G., *Novel resource allocation algorithms to performance and energy efficiency in cloud computing*, The Journal of Supercomputing, vol. 69, no. 3, 2014, 1445–1461.

[6] Xiao Z., Song W., Chen Q., *Dynamic resource allocation using virtual machines for cloud computing environment*, Parallel and Distributed Systems, IEEE Transactions on, vol. 24, no. 6, 2013, 1107–1117.

[7] Schluting C., *Practical VM Architecture: How Do You Scale*, Enterprise networking planet, 2008, http://www.enterprisenetworkingplanet.com/netos/, article.php/3753836/Practical-VM-Architecture-How-Do-You-Scale.htm.

[8] Ellis J., *Thinking Differently about Scalability with Cloud Computing*, VMware Cloud Blog, 2010, http://blogs.vmware.com/vcloud/2010/11/thinking-differently-about-scalability-with-cloud-computing.html.

[9] Telenyk S., Rolik O., Bukasov M., Halusko D., *Models and methods of resource management for VPS hosting*, Technical Transactions, Vol. 4-AC/2013, 41–52.

[10] Telenyk S., *Resource Management for Server Virtualization in the Limitations of Recovery Time Objective*, S. Telenyk, M. Bukasov, M. Yasochka, Proceedings of International Congress on Information Technology, Computational and Experimental Physics (CITCEP'2015), 18–20 December, Cracow 2015, 247–250.