Solving the Virtual Machine Placement Problem as a Multiple Multidimensional Knapsack Problem

Ricardo Stegh Camati, Alcides Calsavara, Luiz Lima Jr.
Programa de Pós-Graduação em Informática – PPGIa
Pontifícia Universidade Católica do Paraná – PUCPR
Curitiba, Brazil
rcamati@ppgia.pucpr.br, alcides@ppgia.pucpr.br, laplimpa@ppgia.pucpr.br

Abstract—Effective placement of virtual machines in a cluster of physical machines is essential for optimizing the use of computational resources and reducing the probability of virtual machine reallocation. Many of previous works treat virtual machine placement as an instance of the bin packing problem, as they aim at saving energy. Alternatively, we propose an approach based on the multiple multidimensional knapsack problem, where the main concern is to maximize placement ratio. Several traditionally employed placement algorithms were re-implemented and new algorithms were defined by using such an approach. The algorithms were evaluated with respect to placement ratio, by employing a novel evaluation method that contemplates variation of computational resources heterogeneity in multiple dimensions and also variation of placement density. The experimental results showed that heterogeneity of resources among physical machines impairs the placement ratio, while heterogeneity of resources among virtual machines benefits it. It was also possible to observe that increase of density placement up to a certain point benefits the placement ratio.

Keywords- cloud computing; virtual machine placement; knapsack problem; evaluation method;

I. INTRODUCTION

The client-server model widely used in modern computing implies that the demand for computer resources (e.g., processing power, memory, storage, and network bandwidth) in a given application can vary significantly over time; the demand may increase or diminish depending on certain days and determined schedules [1]. The cloud computing computation model emerged as a solution to this problem, by permitting *elasticity* of computer resources [2] [10]. Such elasticity is typically implemented through *hardware virtualization* [3], a technique known since the 1960s decade. It appeared with the *time-shared* operating systems [4] [9] and its aim was to partition hardware to isolate users and applications in a *mainframe*. During the 1980s and 1990s decades, hardware virtualization was practically forgotten despite the fact that virtualization was used in other abstractions such as Java Virtual Machine (JVM). More recently, the original concept of virtualization has been employed again, not only to isolate applications and users, but also to permit data centers (large clusters of hosts) to achieve a more dynamic and rapid deployment [5]. Thus, the basic service provided by a data center is the instantiation of virtual machines required by its clients.

There are many benefits for both data center provider and client. In general, a provider is able to sell computer resources to a large number of clients. In addition, the amount of computer resources sold can be higher than the amount actually available, by assuming that, at any given time, the amount of resources actually required by clients is lower than the contracted amount. On the other hand, a client is able to buy computer resources for lower costs, when compared to the costs of keeping a private infrastructure (hardware, software, and personnel). Moreover, since the demands for computer resources may vary over time, a client should pay just for what is actually used, thus reducing costs even further.

The intrinsic complexity of the cloud computing computation model poses a difficult problem regarding resource management. Basically, a data center provider aims at maximizing profits. That implies reducing the deployed computer resources as much as possible, i.e., within each time slot, the amount of computer resources available to clients should be the minimum necessary, according to the corresponding expected demand. However, it should be noticed that the provider must honor the contracts established with clients, whereby quality-of-service requirements, including performance requirements, are specified. In other words, the so-called Service-Level Agreement (SLA) must be fulfilled. As a rule of thumb, the number of hosts (physical machines) in a data center should be augmented gradually, as more client contracts are established. Such a measure for cost reduction is further improved by keeping active only a minimal subset of hosts that fulfills the demand for resources within each time slot, mainly to save energy. In the literature, this problem is referred to as *virtual machine consolidation*.

The critical issues in the design of a virtual machine consolidation mechanism include the choice of algorithms for virtual machine placement and migration. A virtual machine placement is the action of instantiating it as required by a client in a properly chosen host, while a virtual machine migration is the action of moving a virtual machine from one host to another. The reason for migration is that the original host becomes either overloaded or under loaded, according to some criteria. A virtual machine is moved away from an overloaded host to guarantee that computer resource demands and performance requirements are accomplished. On the other hand, it is moved away from an under loaded

 ISBN: 978-1-61208-318-6

host to be able to deactivate such host, thus saving energy. However, migration itself can introduce both some performance degradation and some extra energy consumption, so it should be employed carefully and avoided as much as possible. Unfortunately, virtual machine migration cannot be fully prevented because they are intrinsically dynamic with respect to computer resource demand, a property known as *elasticity*. Nevertheless, a proper initial placement can help to minimize the probability of host overload, consequently reducing migration.

A virtual machine placement algorithm should provide a trade-off between energy savings and overall system performance. If a placement algorithm's objective is solely energy savings, it is likely to happen that hosts become overloaded more often, and clients may experience some performance degradation caused by virtual machine migration. In spite of that, the typical approach employed in real-world data centers is based on the *Bin Packing Problem* and a corresponding rather simple solution: the *First Fit Algorithm*. Within this approach, only the minimal necessary hosts are kept active at all times and only one is candidate (normally, the least occupied host) for virtual machine placement. Some proposals [8] [11] try to improve that solution by reordering the virtual machine request queue according to some criteria before their actual placement; those are referred to as *First Fit Decreasing Algorithms*. Other proposals [16] [17] [18] [19] are based on the *Multiple Knapsack Problem* and a corresponding *Best Fit Algorithm.* Within this approach, hosts are less occupied in average and several of them are the candidates for virtual machine placement at once. As a consequence, less migration is expected at the cost of some extra energy consumption. Also, a new issue emerges regarding the number of computer resource types considered in making a choice between the candidates. Most solutions consider just one resource type, typically processing power. Other solutions consider a combination of resource types, such as processing power, memory, storage, network bandwidth, and so forth. For simplicity, each resource type is referred to as a machine dimension, so such a solution is based on the *Multidimensional Multiple Knapsack Problem*.

Some characteristics of real-world data centers can make the virtual machine placement problem even more complex. One such characteristic is machine heterogeneity, i.e., machines present different amounts for a given dimension. Another characteristic is the average density of virtual machines per host. Such characteristics may profoundly impact on the behaviour of placement algorithms, and that can be very hard to measure.

In this paper, the multidimensional multiple knapsack approach to the virtual machine placement problem is explored in several ways. Firstly, two variants of the First Fit Decreasing Algorithm that are normally employed in the bin packing approach, namely the Dot Product Algorithm and Volume Algorithm, were adapted to the multiple knapsack approach. Secondly, two novel algorithms are proposed, namely the Best Dimension Algorithm and the Osmosis Algorithm. Thirdly, a novel evaluation method for virtual machine placement algorithms is proposed in order to accomplish multiple machine dimensions, heterogeneity of machines and distinct densities of virtual machines per host.

The remaining of this paper is organized as follows. Section II discusses the virtual machine placement problem in deep. Section III describes the normally employed virtual machine placement algorithms. Section IV presents the novel evaluation method. Section V presents some experimental results. Finally, Section VI makes some concluding remarks.

II. PLACEMENT PROBLEM

Given a data center composed of a set of hosts (physical machines) and a corresponding queue of client requests for virtual machine instantiation, the virtual machine placement problem consists in determining a host of the data center to place (actually, instantiate) each virtual machine in the queue, aiming at least at one of the following objectives:

i. To minimize the electric energy consumed by the data center [13] [14]. According to [7], the costs of powering and cooling accounts for 53% of the total operational expenditure of datacenters;
ii. To maximize the placement ratio, i.e., the quotient between the number of placed virtual machines and the total number of requests in the queue;
iii. To minimize the overall systems performance degradation caused by virtual machine migration [6][20] that is triggered by *elasticity*.

The virtual machine placement problem encompasses several issues, as detailed in the sequel.

A. Machine Dimension

A typical virtual machine requires resources of different types, such as processing power (usually measured in MIPS), memory, storage, and network bandwidth [19]. Each resource type is referred to as a *machine dimension*, or simply *dimension*, since it applies to both virtual machine and host. The demand in a certain dimension can vary over time within each virtual machine, and it not known a priori. For simplicity, it is assumed that a demand initially defined is actually an upper bound. Naturally, a virtual machine must be placed in a host that can provide enough resources in all dimensions. However, the criterion to choose a host where to place a virtual machine can be based on either a single dimension or a combination of any number of dimensions. For instance, one may decide according to processing power only, such that a virtual machine is placed in the host where the processing power dimension is mostly available, as long as the resources available in the other dimensions are enough too. As another example, one may make a placement decision based on some ranking which is determined according to the available resources in the above four dimensions. While a single-dimension criterion is simple to implement, a multi-dimension criterion may help to increase the balance between resource usages, and placement ratio as well.

 ISBN: 978-1-61208-318-6

B. Delivery Mode

Although requests for virtual machine placement arrive continuously at a data center, their corresponding delivery, that is, their actual placement on hosts can be performed in two distinct modes: *(i) single delivery mode*: the first virtual machine in the queue is popped out and placed on a host, that is, virtual machines are necessarily delivered in the same order of arrival of requests; *(ii) group delivery mode*: a number of virtual machine requests in the beginning of the queue are popped out and placed on a host. That allows reordering the group of virtual machines before placement. The single delivery mode is simple to implement, it respects requests arrival order, and it causes no additional delay in virtual machine placement delivery. The group delivery mode, on the other hand, has a potential to increase the virtual machine placement ratio since virtual machine requests can be reordered in an attempt to optimize resource usage.

C. Heterogeneity

The set of hosts in a data center can vary with respect to the resources they provide. In the same fashion, the set of virtual machines to place can vary with respect to the resources they require. In other words, both set of hosts and set of virtual machines are heterogeneous with respect to machine dimensions. Such heterogeneity adds complexity to the virtual machine placement problem. On the other hand, it can be exploited to reach the defined objectives.

D. Virtual Machine Life Cycle

Once a virtual machine is instantiated (placed on an initial host), it can change its resource demands over time (*elasticity)*, it can migrate from one host to another, and it terminates eventually. Such behavior may affect the system's performance, energy consumption and resource availability. Hence, it has to be considered during virtual machine placement. It should be noticed that virtual machines are not pre-allocated, i.e., each instantiation request implies creating a new virtual machine.

III. PLACEMENT ALGORITHMS

In this section, traditionally employed virtual machine placement algorithms are explained, along with two new such algorithms are proposed, namely the *Best Dimension Algorithm* and the *Osmosis Algorithm*.

A. First Fit Algorithm

The classic *Bin Packing Problem* [11] is often adapted to the context of virtual machine placement in the following way: a whole set of virtual machines (objects) of different sizes should be placed into a series of hosts (bins) such that the minimum number of hosts are employed to place all virtual machines, thus saving energy. The First Fit Algorithm [8][11] accomplishes that by activating a single host at a time as they get filled up with virtual machines. Each virtual machine is placed in the *first* host where it *fits*, according to a predefined order between active hosts. In addition, just one virtual machine – the first one in the queue – is taken at a time, and then placed on a host. For that reason, the First Fit Algorithm is suitable for the single delivery mode.

B. First Fit Decreasing Algorithm

When the group delivery mode (as described in Section II.B) is employed, the First Fit Decreasing Algorithm [8][11] – a variant of the First Fit Algorithm – can reduce even further the average number of active hosts. The primary difference is that the set of virtual machines in the queue is ordered according to some criteria before placement. In a simple model, where just one dimension is considered (typically, CPU usage), virtual machines are ordered according to their demand for the corresponding resource. In a more sophisticated model, several dimensions can be considered to establish a ranking amongst the virtual machines. One way to defining the rank of each virtual machine is to employ the *volume method* whereby the volume of a virtual machine is calculated by multiplying its demands in all dimensions.

In a different approach, the rank of each virtual machine can be determined with respect to a reference host witch is, normally, the least occupied active host or the last one to be activated. In this approach, at least during rank calculation, the virtual machine placement problem can be modeled as an instance of the *0-1, or Binary, Knapsack Problem* [12], as follows. Given a set (group) of virtual machines (objects) where each virtual machine has an associated value, a subset must be selected and associated to a single host (knapsack) such that the *profit* is maximized. An example of an algorithm that takes this approach is one that employs the *dot product* method. The *dot product* of a virtual machine is calculated as the sum of a series of products, where each product corresponds to a dimension, and is obtained by multiplying the virtual machine demand by the reference host availability in that dimension.

For simplicity, the First Fit Decreasing Algorithm based on the volume method is referred to as *Volume Algorithm* [15], while the First Fit Decreasing Algorithm based on the dot product method is referred to as *Dot Product Algorithm* [8].

C. Best Fit Algorithm

Besides saving energy, a data center should to take measures to fulfill all Service-Level Agreement (SLA) requirements, including virtual machine placement ratio (which is equivalent to establishing a maximum time for virtual machine placement), virtual machine performance and virtual machine elasticity.

The Best Fit Algorithm keeps active a set of hosts at all times (instead of activating a single host at time) and places each virtual machine where it best fits to achieve load balance amongst the active hosts. A proper load balance should increase the probability of success in virtual machine

 ISBN: 978-1-61208-318-6

placements. Moreover, it should increase the probability of having enough resources at a given host to fulfill the needs due to a virtual machine expansion. As a consequence, virtual machine migration should be reduced, thus preventing systems performance degradation.

Basically, the problem solved by the Best Fit Algorithm can be seen as an instance of the *Multiple Knapsack Problem*: given an initial set of virtual machines (objects) of different sizes and values, a subset of it should be selected, and then placed into a set of hosts (knapsacks) such that the aggregate value is maximized. Moreover, when multiple dimensions are considered, the problem is an instance of the *Multiple Multidimensional Knapsack Problem* [16][17][18][19]. Particularly, if all virtual machines are assumed to hold a unique value, the Best Fit Algorithm maximizes the quantity of virtual machines that are placed. The remaining issue concerns the delivery mode, as discussed in the sequel.

1) Single delivery mode

Since just one virtual machine – the first one in the queue – is taken at a time, and then placed on a host, there is no actual virtual machine selection in the single delivery mode. Such a reference virtual machine will be simply either placed or discarded (actually, it can be reinserted at the end of the queue), depending on the resource availability on the active hosts. Nevertheless, the analogy with the Multiple Knapsack Problem holds if the whole queue of virtual machines (built within a certain period of time) is considered as the initial set of objects. In this case, the selected virtual machines are the ones that are placed, i.e., the ones that are not discarded. Thus, the purpose of the Best Fit Algorithm is to discard virtual machines the least as possible. In other words, the purpose is to maximize placement ratio in the long run. The actual issue, hence, is to determine the host, if there is any, where the reference virtual machine fits best. That requires ordering the set of active hosts according to some criteria, before placing the reference virtual machine. Similar to the First Fit Decreasing Algorithm (described in Section III.B), both *volume* and *dot product* methods can be employed in this case. If the volume technique is employed, host are sorted in decreasing order by their *free volume*, which is calculated as the product of available resource in each dimension. If the dot product is employed, the rank of each host is calculated as the sum of a series of products, where each product corresponds to a dimension, and is obtained by multiplying the reference virtual machine demand by the host availability in that dimension. A slightly different version of the dot product method is the *best dimension* method, proposed here, by which the product for each dimension is calculated in the same fashion as in the dot product method, but the rank of each host is set simply as the highest product. For simplicity, the Best Fit Algorithm based on the best dimension method is referred to as *Best Dimension Algorithm*.

2) Group delivery mode

The analogy with the Multiple Knapsack Problem is direct in the case of the group delivery mode, since it permits a selection of the virtual machines that best fit the set of active hosts, before placement. Actually, there are two kinds of selection that can be exploited in any form by a placement algorithm: a selection of virtual machines and a selection of active hosts.

An algorithm that employs only a virtual machine selection method should order the set of virtual machines according to some criteria, and then place each virtual machine in any active host. For example, an algorithm may order the virtual machines according to their required processing power. If more dimensions should be considered, they can be ordered according to the required *volume*.

An algorithm that employs only a host selection method should order the set of active hosts according to some criteria before placing each virtual machine that is simply popped out from the queue. For example, the set of active hosts can be ordered according to the proximity between their level of free processing power and the average processing power required by the all virtual machines in the group. It should be noticed that such an algorithm differs from an algorithm employed for the single delivery mode because its host selection method uses attribute values of all virtual machines in the group, instead of attribute values of just the first virtual machine in the queue.

An algorithm that employs both selection methods simultaneously actually employs a *match method* between virtual machines and active hosts. For example, a simple match method is to associate virtual machines that require higher processing power with hosts with higher level of free processing power.

The *Osmosis Algorithm*, proposed here, employs the host selection method only, albeit it can be easily extended to employ the virtual machine selection method as well is straightforward. The Osmosis Algorithm attempts to preserve resources that are scarce in the data center with respect to the current demand. Its host selection method consists in ordering the set of active hosts according to their availability of relatively least available resources. Hosts that hold resources whose usage levels are less critical in the data center are used first. The first step is to determine which resources present more critical levels of usage. That is achieved by determining the *weight* of each dimension as the quotient between its total demand and its total availability. The *total demand* of a dimension is defined by the sum of all demands required the virtual machines in the group, while the *total availability* of a dimension is the sum of free resource in all the active hosts. Once the weight of each dimension has been determined, the rank of each active host must be calculated with respect to the first virtual machine in the queue – the reference virtual machine – as follows. For each host, the *weighted offer ratio* of a dimension is defined as the product between the weight of that dimension and the corresponding *offer ratio,* which is

 ISBN: 978-1-61208-318-6

defined as the quotient between the host availability and the reference virtual machine demand for that dimension. Finally, the rank of each host is calculated as the sum of its weighted offer ratios, in all dimensions.

D. Discussion

The First Fit Algorithm and its variants make it possible for a data center to save energy, since only hosts that contain some virtual machine need to be on. However, because host resource usage levels tend to be close to the maximum, virtual machine migration is more likely to happen in the presence of elasticity (when, within an already running virtual machine, the demand for a given resource increases), thus degrading performance, besides consuming some extra energy.

On the other hand, the Best Fit Algorithm and its variants should improve placement ratio, which brings benefits for both data center provider and client, thus justifying the extra energy cost. In addition, virtual machine migration caused by host overload is less likely to happen, hence improving the overall systems performance.

In the following sections, an evaluation of the different placement algorithms, with respect to placement ratio, is presented, while evaluation of issues regarding elasticity, migration, performance and energy consumption are left as future work.

IV. EVALUATION METHOD

A new method to evaluate virtual machine placement algorithms is proposed here in order to take into account multidimensional machines, and to investigate the impact of machine heterogeneity and virtual machine density on placement ratio. Basically, the method consists in determining a set of virtual machines to place on a set of hosts, and applying several placement algorithms in order to compare their behavior with respect to placement ratio. Every machine (either host or virtual machine) is assumed to have a number of dimensions, where each dimension corresponds to a certain type of resource. The set of hosts is assumed to be heterogeneous with respect to the resource capacity and, for each dimension, each host may have a different capacity. In the same way, the set of virtual machines should present a certain degree of heterogeneity.

The virtual machine placement problem can be formalized as follows.

Be:

- $H = \{h_1,...,h_m\}$: *a set of m hosts*
- $V = \{v_1,...,v_n\}$: *a set of n virtual machines*
- $D = \{d_1,...,d_g\}$: *a set of g dimensions*
- T_j : *the arrival time of* $v_j \in V$ *such that* $\forall v_i, v_j : \; i < j \;\Leftrightarrow\; T_i < T_j$
- $w_{j,k}$: *the demand of* $v_j \in V$ *in dimension k*
- $c_{i,k}$: *the total capacity of* $h_i \in H$ *in dimension k*
- $L_{i,k}(T_j)$: *the free capacity of dimension k in* $h_i \in H$ *at* T_j
- $x_{i,j} \in \{0,1\}, \quad 1 \le i \le m, \quad 1 \le j \le n$
  - $x_{i,j} = 1 \Leftrightarrow v_j \in V$ *is placed in* $h_i \in H$
  - $x_{i,j} = 1 \Rightarrow x_{k,j} = 0, \forall k \ne i$

Maximize $\sum_{i=1}^{m}\sum_{j=1}^{n} x_{i,j}$

Subject to:

(i) $\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{g} x_{i,j}.w_{j,k} \le c_{i,j}$

(ii) $\forall v_j \in V, \;\; \exists \;\; S \subseteq H, \;\; S \ne \varnothing \;\; | \;\; \forall h_i \in S, \;\; \forall d_k \in D, \;\; L_{i,k}(T_j) \;\; \ge \;\; w_{j,k} \;\; \Rightarrow \;\; x_{t,j} = 1, \;\; h_t \in s$

The analysis of the effects of heterogeneity on placement ratio employs an *ideal scenario* as a starting point. In such a scenario, hosts and virtual machines are assumed to be homogeneous, and their capacities are such that all virtual machines are placed and, also, no resource is left in any host. For a given set of hosts, the number of virtual machines is determined by according to an arbitrarily fixed virtual machine density; if distinct values of virtual machine density are employed, then the number of virtual machines is determined accordingly. In the same fashion, for each machine dimension, the corresponding capacity in each host and the demand of each virtual machine are determined according to virtual machine density. Hence, the ideal scenario for a given set of hosts and a given virtual machine density includes the number of virtual machines to place, and the capacity of hosts and the demand of virtual machines for each machine dimension.

After having established an ideal scenario, other scenarios can be devised by introducing some degree of heterogeneity in both set of hosts and set of virtual machines. In such a scenario, for any given dimension and a set of machines, there is a specific range that includes all corresponding dimension values. A range can be defined by a minimum, a maximum and a corresponding median that is determined according to the ideal scenario (host capacity or virtual machine demand). The relative variation of dimension values with respect to the median is called the

 ISBN: 978-1-61208-318-6

amplitude of that dimension with respect to that set of machines.

Formally, given a range *[minimum, maximum]* of values of a dimension *d* in a set of *M* of machines, the *amplitude* of *d* with respect to *M* – denoted as $A_{M,d}$ – is defined in (1).

$$A_{M,d} = 100\ (maximum-median)/median \quad (1)$$

In a set of *M* of machines where each machine has *n* dimensions that are labeled from *1* to *n*, the tuple containing the amplitudes for all dimensions – denoted as $A^*_{M,d}$ – is defined in (2).

$$A^*_{M,d} = (A_{M,1},...,A_{M,n}) \quad (2)$$

Given a set *V* of virtual machines that should be placed, let $P \subseteq V$ be the set of virtual machines actually placed after applying a certain algorithm, *v* be the number of machines in *V*, and *p* be the number of machines in *P*. The corresponding *placement ratio* -- denoted as τ -- is defined in (3).

$$\tau = \frac{p}{v} \quad (3)$$

The *ideal scenario* is one such that $\tau = 1$, i.e., the whole set of virtual machines is successfully placed, and also all host resources are fully occupied. Such a scenario can be easily synthesized when there are *v* virtual machines to place on *h* hosts, by assuming that:

(i) All amplitude values are *zero*, for both hosts and virtual machines. In other words, hosts and virtual machines are homogeneous.

(ii) For each machine dimension, let *k* be the corresponding capacity in each host, and *t* be the corresponding demand of each virtual machine. Then, $k \bmod t = 0$ (there is no resource left in each host) and $h \cdot k = v \cdot t$ (there is no resource left in the data center).

Given a set of *h* hosts and a set of *v* virtual machines, the average number of virtual machines that should be placed per host – denoted as ρ – is simply defined in (4).

$$\rho = \frac{v}{h} \quad (4)$$

In the ideal scenario, $\rho = \frac{k}{t}$ for any machine dimension.

V. EXPERIMENTS

The algorithms described in Section III were experimentally evaluated with respect to placement ratio, according to the method described in Section IV. The experiments were based on simulation by employing a custom simulator (available for download at http://www.ppgia.pucpr.br/~alcides/PSim) written in Python language, and they were divided into seven scenarios that correspond to distinct degrees of machine heterogeneity, and distinct densities of virtual machines, as well. The number of hosts is fixed to 100 for all scenarios, except for Scenario VI, where this number is ten due to simulation time constraints. For most scenarios, virtual machine density is fixed to ten virtual machines per host, which is a typical density for small instances in data centers. The number of dimensions is fixed to four since the most commonly considered resources are processing power, memory, storage, and network bandwidth. Consequently, the amplitude tuples contain four values corresponding to heterogeneity of those resources. For simplicity, in the experimental results shown above, a host amplitude tuple is denoted as δ, while a virtual machine amplitude tuple is denoted as π.

In scenarios I and II, host amplitude is fixed in order to analyze the impact of the virtual machine amplitude variation, while, in scenarios III and IV, virtual machine amplitude is fixed in order to analyze the impact of the host amplitude variation. Also, in scenarios I and III, the machine amplitude variation increases evenly, while it increases non-uniformly in scenarios II and IV.

According to Fig. 1 and Fig. 2, increasing virtual machine amplitude either evenly or non-uniformly favors all algorithms. The Dot Product Algorithm was the best performer, achieving a placement ratio from 2% to 10% better than the First Fit Algorithm.

According to Fig. 3 and Fig. 4, increasing the host amplitude variation either evenly or non-uniformly impairs the performance of all algorithms. The Dot Product Algorithm shows the best overall performance, achieving from 6% to 19% more performance than the First Fit Algorithm.

In scenarios V and VI, the impact of virtual machine density is analyzed. In this case, host amplitude and virtual machine amplitude are fixed. While Scenario V considers virtual machines that have a monolithic operation system as guest, the Scenario VI considers very small virtual machines that have a microkernel operational system as guest [21]. The results shown in Fig. 5 are quite interesting. Increasing the virtual machine density favors all algorithms. The Osmosis Algorithm was the best solution for very small virtual machine density (from 2 to 3), performing from 5% to 9% better than the Volume Algorithm, the worst performer. With a density higher than 3, once more the Dot Product Algorithm was the best performer, achieving a placement ratio from 3% to 12% higher than the First Fist, the worst performer in this range.

 ISBN: 978-1-61208-318-6

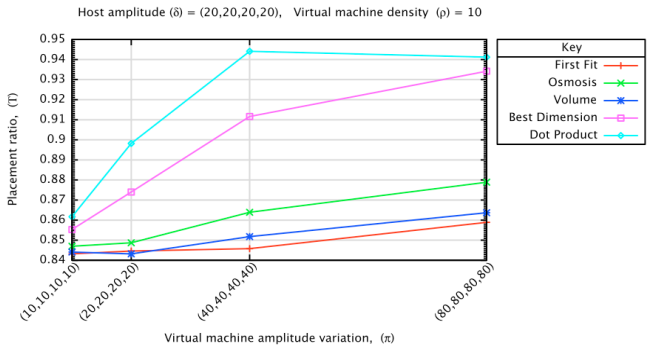

Figure 1. Increasing virtual machine amplitude variation evenly.

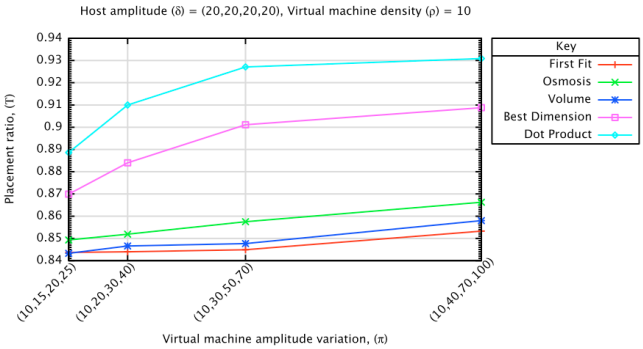

Figure 2. Increasing virtual machine amplitude variation non-uniformly.

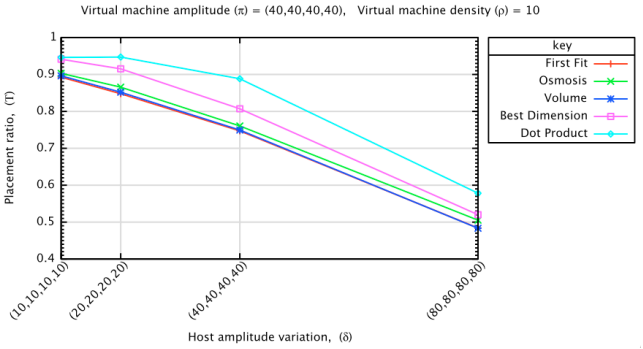

Figure 3. Increasing host amplitude variation evenly.

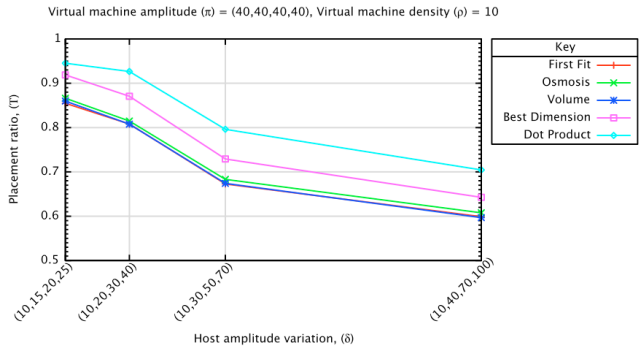

Figure 4. Increasing host amplitude variation non-uniformly.

According to Fig. 6, increasing density is beneficial for all algorithms, at least up to a certain point. As expected, the Dot Product Algorithm proved to be superior to all other algorithms, being from 6% to 9% more efficient than the First Fit Algorithm and the Volume Algorithm.

In Scenario VII, placement ration of the Dot Product Algorithm is analyzed by varying both host amplitude and virtual machine amplitude, simultaneously. The choice of the algorithm is due to its higher performance, as verified in previous scenarios. In this scenario, virtual machine density is fixed as 10, as well. In Fig. 7, it can be noticed there are several depressions caused by increasing host amplitude variation and also by the decreasing virtual machine amplitude variation. It can be noticed that virtual machine heterogeneity tends to improve placement ratio, while host heterogeneity tends to degrade it.

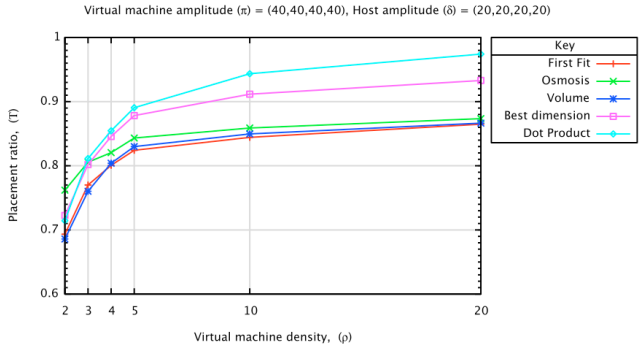

Figure 5. Increasing virtual machine density in a monolithic approach.

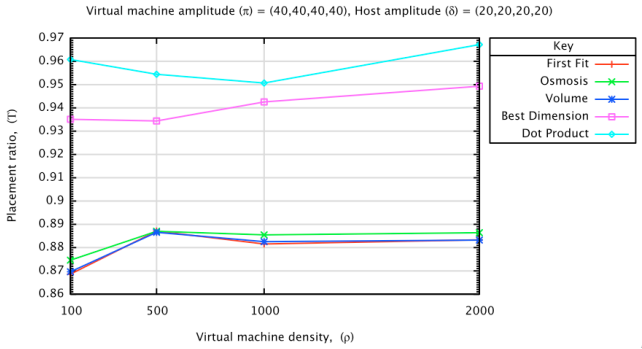

Figure 6. Increasing virtual machine density in a microkernel approach.

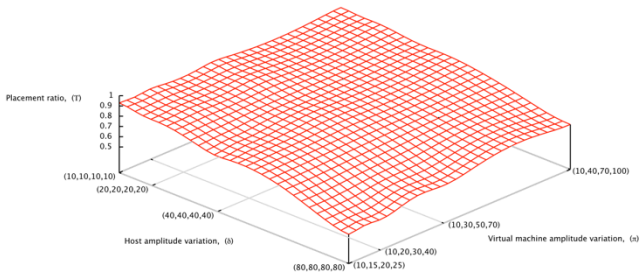

Figure 7. Increasing both host amplitude variation evenly and virtual machine amplitude variation non-uniformly

 ISBN: 978-1-61208-318-6

In conclusion, the Dot Product Algorithm seems to be the best solution in most cases. The Best Dimension Algorithm and the Osmosis Algorithm, proposed in this paper, showed a median performance. While Best Dimension has a better overall performance than Osmosis, Osmosis is the best choice in the case of very low virtual machine density, surpassing even the Dot Product Algorithm. A real world scenario with 2 or 3 virtual machines per host is a very common with large instances, as an example a LAMP (Linux, Apache, MySQL, PHP) Application. The First Fit Algorithm and the Volume Algorithm are very low performers. On the other hand, as they are simple to implement, they can be appropriate when there is a performance bottleneck in the virtual machine placement system.

VI. CONCLUSION

This work has shown that the multidimensional multiple knapsack approach can be more effective than the bin packing approach to the virtual machine placement problem; in general, the First Fit Algorithm was the worst performer in our experiments. The proposed evaluation method permitted to represent critical real-world data center characteristic, and it proved to be easy to use. The experiments have shown that virtual machine heterogeneity plays a favorable role in virtual machine placement, while host heterogeneity has the opposite effect. Also, in general, high virtual machine density favors virtual machine placement ratio. Finally, algorithms that use properties of virtual machines (Dot Product, Best Dimension and Osmosis Algorithm) tend to have a better performance.

The proposed evaluation method can be extended to include energy consumption, elasticity, migration, and termination. More future works include the evaluation of other types (genetic, ant colony, etc.) of placement algorithms, and experimentation with more fixed values for virtual machine density. Also, the costs of running each placement algorithm should be computed to verify its feasibility in large-scale scenarios, and to verify if the corresponding gain in placement ratio is worthwhile. Other issues regarding SLA, such as services response time and security are left as future work as well.

REFERENCES

[1] H. Liu and S. Wee, “Web Server Famrm in Cloud: Performance Evaluation and Dynamic Architecture”, Proceedings of the First International Conference on Cloud Computing, Technology and Science, Berlin and Heidelberg, Springer Verlag, 2009, pp. 369-380.

[2] P. Mell and T. Grance, “The NIST Definition of Cloud Computing”, Technical Report, Gaithersburg, 2011, National Institute of Technology Special Publication 800 – 145.

[3] E. J. Smith and R. Nair, “The Architecture of Virtual Machines”, Computer, vol.38(5), May. 2005, pp. 32-38.

[4] V. Melinda, “VM and the VM Community: Past, Present and Future”, Office of Computing and Information Technology, Princeton University, Share 89 Sessions 9059-9061, 1997, pp. 1-68.

[5] M. Rosenblum, “The Reincarnation of Virtual Machines. Virtual Machines”, Virtual Machines, vol.2(5), Aug. 2004, pp. 34-40.

[6] M. R. Hinnes, U. Deshpande, and K. Gopalan, “Post Copy Live Migration of Virtual Machines”, ACM SIGOPS Operating System Review, vol.43(3), 2009, pp. 14-26.

[7] Q. Zhang, L. Cheng, and R. Boutaba, “Cloud Computing: State of Art and Research Challenges”, Journal of Internet Services and Applications,vol. 1(1), Apr. 2010, pp. 17-18.

[8] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wider, “Heuristics for Vector Bin Packing”, Microsoft’s VMM Product Group, Microsoft Research Sillicon Valley, 2011.

[9] R. J. Creasy, “The Origin of VM/370 Time-Sharing System”, IBM Journal of Research and Development, vol. 25(5), pp. 483-490.

[10] T. P. Endo, E. G. Gonçalves, J. Kelner, and H. D. F. Sadok, “A Survey on Open-Source Cloud Computing Solutions”, Proceedings of the XXVII Brazilian Simposium of Computer Networks and Distribuited Systems, VII Workshop in Cloud Computing, Porto Alegre, Brazil, 2010, pp. 3-16.

[11] B. Xia and Z. Tan, “Tigther bounds of the First Fit Algorithm for the Bin-Packing Problem”, Elsevier, Hangzhou, vol. 158(15), Aug. 2010, pp. 1668-1675.

[12] D. Pisinger, “Algorithms for Knapsack Problems”, Department of Computer Science of University of Copenhagen, Copenhagen, Feb. 1995.

[13] Y. Wu, M. Tang, and W. Fraser, “A Simulated Annealing Algorithm for Energy Efficient Virtual Machine Placement”, Proceedings of IEEE International Conference on Systems, Man and Cybernetics, Seoul, Oct. 2012, pp. 14-17.

[14] J. Xu and J. A. B. Fortes, “Multi-Objective Virtual Machine Placement in Virtualized Data Center Enviroments”, Proceedings of the 2010 IEEE/ACM Int’l Conference on Green Computing and Communications & Int’l Conference on Cyber, Pysical and Social Computing, Washington, 2010, pp. 179-188.

[15] D. Bonde, “Techniques for Virtual Machine Placement in Clouds”, Departament of Computer Science and Engineering - MPT Stage I Report on the Indian Institute of Technology, Bombay, 2010, ROLL No:08305910.

[16] S. Fidanova, “Heuristics for Multiple Knapsacks Problem”, Proceeding of the IADIS International Conference on Applied Computing, Algarve, 2005, pp. 225-260.

[17] Y. Song, C. Zhang, and Y. Fang, “Multiple Multidimensional Knapsack Problem and it’s Applications in Cognitive Radio Networks”, Military Communications Conference, San Diego, Nov. 2008, pp. 1-7.

[18] A. Singh, M. Korupolu, and D. Mohapata, Server-Storage Virtualization: Integration and Load Balance in Data Centers, International Conference for High performance Computing, Network, Storage and Analyses, pp. 1-12.

[19] E. Mohamadi, M. Karimi, and S. R. Heikalabad, “A Novel Virtual Placement in Virtual Computing”, Australian Journal of Basic and Applied Scienes, Australia, vol. 5(10), 2011, pp. 1149-1555.

[20] D. Magalhães, J. M. Soares, and D. G. Gomes, “Virtual Machine Migration Impact Analysis in a Virtualized Computer Enviroment”, Proceedings of the XXIX Brazilian Symposium on Computer Networks and Distributed Systems, Campo Grande, 2011, pp. 235-248.

[21] A. Whiteaker, M. Shaw, and S. T. Gribble, “Denali: A Scalable Isolation Kernel”, Proceedings of the 10th Workshop on ACM SIGOPS European Workshop, New York, 2002, pp. 10-15.

 ISBN: 978-1-61208-318-6