Two are better than one: An algorithm portfolio approach to cloud resource management

Zoltán Ádám Mann

paluno – The Ruhr Institute for Software Technology
University of Duisburg-Essen, Essen, Germany

Abstract. Several different algorithms have been proposed in recent years for the dynamic optimization of resource allocation in virtualized data centers. The proposed methods range from fast and simple heuristics to exact algorithms that yield optimal results but take much longer. This paper suggests an algorithm portfolio approach in which multiple algorithms coexist. Based on continual monitoring and analysis of the state of the data center, the optimization algorithm that is most suitable is chosen on the fly. This way, the balance between optimization quality and reaction time can be tuned adaptively. Empirical results show that this approach leads to improved overall results.

This paper has been published in the Proceedings of the 6th European Conference on Service-Oriented and Cloud Computing (ESOCC 2017), Springer LNCS volume 10465, pp. 93-108, 2017.

1 Introduction

The last years have witnessed a tremendous uptake of cloud computing. The compelling advantages of the cloud, like the instantaneous access to services without the need for upfront investments and the elastic scaling backed by a seemingly unlimited pool of resources continue to drive ever more customers to the cloud.

For a provider of Infrastructure-as-a-Service (IaaS), several important challenges must be addressed to provide the service economically and in good quality [10]. First, the operation of the physical infrastructure is associated with high costs. Especially the costs for electricity play an important role for operating servers and cooling equipment [8]. For this reason, virtualization is widely used to achieve high utilization of physical servers and switch off unused ones. In particular, live migration of virtual machines (VMs) between physical machines (PMs) makes it possible to react to changes in the workload and continually consolidate VMs to just the required number of PMs [35].

Second, customers require a high level of service quality. In the case of IaaS, the most important quality objective is that the amount of resources requested for a VM should be available whenever the application in the VM requires it. This objective of the customers is in conflict with the economic objective of the providers. The latter would dictate aggressive consolidation of VMs, but if the resources of a PM are over-subscribed by multiple VMs and the load of the VMs starts to rise, this can quickly lead to an overload of the physical resources, resulting in a situation where VMs do not obtain the requested amount of resources. This may lead to degraded performance for client applications, thus to customer dissatisfaction which may manifest itself in penalties (if the service level agreement mandates this) or customer churn.[1]

As can be seen, it is vital for the provider to find the right balance between the conflicting objectives of minimizing the number of used PMs and minimizing the situations where a PM is overloaded. This leads to an interesting optimization problem called the *VM consolidation problem* [24]. Most of the realistic formulations of the VM consolidation problem are NP-hard to solve optimally or even to approximate with low approximation factors [23]. Still, because of its practical relevance, many algorithms have been proposed to solve this problem.

Many of the suggested algorithms are greedy heuristics that deliver a solution very quickly. However, there is no guarantee on how close the found solution will be to the optimum and in unfortunate cases, it can be very far from it. On the other extreme, some researchers have also proposed exact algorithms that are guaranteed to find the optimum, although at the cost of exponential execution times. To be practical, such algorithms must be furnished with a timeout so that overly long runs are prohibited (in which case the algorithm returns the best solution it has found). This way, the found solution is not guaranteed to be optimal; however, experience shows that this way significantly better results can be achieved than with the simple greedy heuristics, although with also significantly higher execution time.

It is not clear which of these approaches is the most appropriate. For example, in a situation where the workload is quickly rising (e.g., as a result of the flash crowd phenomenon [29]), it is paramount to react quickly. In this case, a greedy algorithm that delivers a suboptimal result within a second is clearly preferred over a more sophisticated algorithm that would give a better result after a minute because by that time PMs may already be overloaded. On the other hand, in a peaceful period of low load, it would pay off to wait for the better allocation returned by the longer-running algorithm.

Based on these considerations, we propose here an *algorithm portfolio* approach, in which the provider has a set of algorithms at its disposal and chooses from them dynamically, based on the current situation of the cloud. This way, the strengths of different algorithms can be combined.

In this paper, we describe a general approach for using an algorithm portfolio for VM allocation, as well as a specific preliminary implementation using two algorithms. Empirical results show that already our preliminary implementation leads to better results than those of the individual algorithms.

[1] Beyond these two basic objectives, there can be also several other factors that the provider must take care of, such as security and privacy requirements, optimization of data transfer among the VMs, thermal issues etc.

2 Previous work

The VM consolidation problem has received a lot of attention in recent years. Several different versions of the problem have been studied and many different algorithms have been proposed to solve it [22, 26]. The proposed algorithms realize different trade-offs between solution quality and algorithm execution time.

The fastest algorithms are greedy heuristics: their running time is at most quadratic in the size of the problem instance, leading to very low execution times, but also to solutions, the quality of which may not be so good. Typical examples include the packing heuristics adopted from the related bin-packing problem, such as First-Fit, Best-Fit, First-Fit-Decreasing etc. [4, 5, 13, 17, 20, 30, 36, 37] and also some proprietary methods [4, 32, 33, 38, 39].

Exact methods (i.e., algorithms that are guaranteed to yield optimal results) are the other extreme. The proposed exact algorithms rely almost always on some form of mathematic programming (e.g., integer linear programming) and appropriate solvers [13, 14, 25, 31, 41]. Unfortunately, these approaches do not scale to practical problem sizes, so their running has to be limited.

There are also some further algorithms. These include meta-heuristics, the execution time and quality of which can be tuned with multiple parameters, e.g., simulated annealing [16, 27], genetic algorithms [11], particle swarm optimization [18], ant colony optimization [9], and biogeography-based optimization [19, 42]. Also, some complex proprietary heuristics fall into this category [2, 17, 28].

The algorithm portfolio approach advocated in this paper was originally suggested by Huberman et al. [15] and then popularized in the artificial intelligence community by Gomes and Selman [12] with the aim of attacking hard combinatorial problems. The fundamental idea is to select from a pool of available algorithms the most appropriate one for each specific problem instance, based on quickly computable features of the problem instance and a model of expected behavior of the algorithms on problem instances with the given features. The most well-known application of this approach has been the SATzilla solver for the Boolean satisfiability (SAT) problem [40], which has consistently achieved top results in the SAT competitions. The approach has also been used in the context of automated synthesis and deployment of cloud applications [7, 1].

3 General problem description

As shown in Fig. 1, the inputs to the VM consolidation problem consist of (i) information about the PMs, (ii) information about the VMs, (iii) the current mapping of VMs on PMs, and (iv) further constraints.

Each PM is characterized by its capacity, current state, and its power consumption characteristic. The capacity can be one-dimensional if only a single resource type (typically the CPU) is considered, or multi-dimensional if multiple resources types (e.g., CPU, RAM, disk) are taken into account. The state of the PM can be either "on" or "off". The power consumption characteristic of the PM is a function that defines how much power the PM consumes depending on the load of the PM.

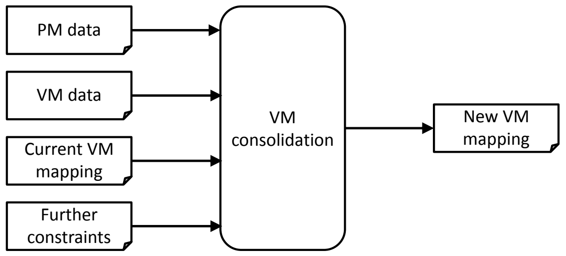

Fig. 1. Inputs and outputs of VM consolidation

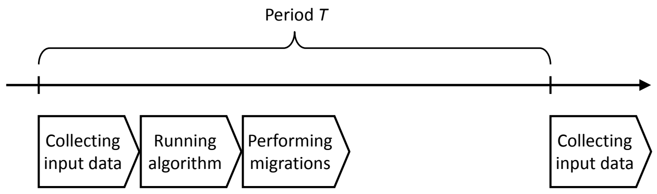

Fig. 2. Time-line of VM consolidation

A VM is characterized by its resource requirements. If d resource types are considered for the capacity of PMs, then also the resource requirements of the VMs are d-dimensional.

Some of the VMs may already exist and be placed on a PM. There can also be newly requested VMs whose placement is not decided yet. Hence, the current mapping of VMs on PMs defines for a subset of the VMs on which PM they currently reside. Further, it is also possible that the termination of some VMs has been requested; such VMs also appear in the current mapping of VMs on PMs, but can be removed.

There can also be further constraints that VM consolidation has to respect. For example, anti-colocation constraints prescribe that certain pairs of VMs must not be placed on the same PM for reasons of security or fault tolerance.

The aim of VM consolidation is to determine a new mapping of VMs on PMs. This mapping must define for each VM – including both existing and newly requested VMs – the PM that should host it. For the newly requested VMs, the new mapping defines on which PM they should be deployed. For existing VMs, if the new mapping defines a different host from the current one, then a migration must be carried out; otherwise, no action is required.

VM consolidation has two main objectives: (i) minimizing total energy consumption and (ii) minimizing PM overloads. These two objectives are conflicting: minimizing energy consumption can be achieved by aggressively consolidating the VMs to as few highly loaded PMs as possible, but this would increase the probability of PM overloads. Therefore, the aim is to find a good balance between these two objectives.

Timing also plays an important role in VM consolidation. The workload keeps changing, and so the mapping of VMs on PMs should be re-optimized regularly to react to the changes. Fig. 2 depicts a typical time-line. According to this, VM consolidation is carried out periodically, with a period of T. In each period, first the input data – in particular, the current load of the VMs – are collected, which are then fed into the consolidation algorithm. Finally, the migrations that the algorithm decided are executed.

Collection of input data can be done in a decentralized manner and hence in parallel, so that the time required for that is not so high. In contrast, the time for running the algorithm can be substantial depending on the specific algorithm used. Also the migrations can take long depending on several factors like memory size of the migrated VMs or the available network bandwidth [34].

The time that elapses between collecting the input data and reaching the new state is critical for two reasons. First, the more time passes, the less effective is the reaction of the system: in case of a PM overload, it takes longer to remedy the problem; if there are consolidation opportunities, it takes longer to exploit them, thereby wasting energy. Second, the workload also changes during this time, so that the state actually reached will be different from the one that the algorithm determined based on the old load levels, and the longer it takes to reach the new state, the higher the difference can be.

For these reasons, the usefulness of a VM consolidation algorithm not only depends on how well it can consolidate the VMs and how well it can eliminate PM overloads, but also how fast it is. The algorithms that have been proposed so far in the literature differ strongly along these dimensions: some are slow but deliver very good results, whereas others are much faster but deliver weaker results. The question that we are trying to address is how the complementary strengths of existing algorithms can be combined.

4 Proposed approach

An overview of our proposed approach is sketched in Fig. 3. The main idea is to use multiple VM consolidation algorithms that offer different trade-offs between speed and quality. In each period, it is decided dynamically which of the available algorithms should be used in the current optimization period. This decision is based on a quick analysis of the current system state, consulting a knowledge base containing information about the characteristics of the available algorithms. The analysis has to be quick because it is on the critical path of the decision-making process. Hence it should consist of simple rules that are based on aggregate system metrics. For example, such a rule could state that the fastest algorithm should be chosen if there are several overloaded PMs or if violations of some security-related constraints have been detected.

Some algorithms have important parameters with which their behavior can be configured. Some parameters may relate to quantities of the problem domain; for instance, some algorithms support explicit thresholds on the number of migrations [3] or the headroom to leave on PMs to prevent overloads [5]. Other

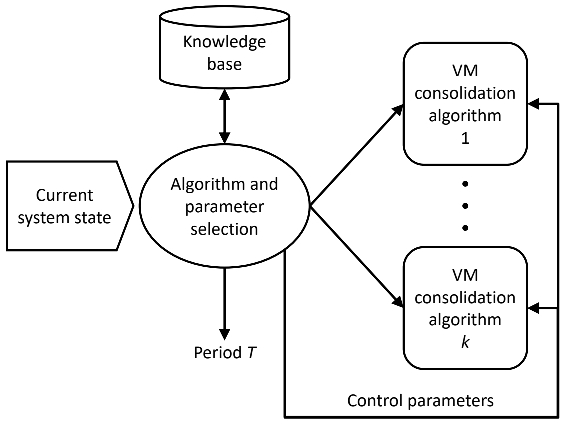
Knowledge base
Current system state
Algorithm and parameter selection
VM consolidation algorithm 1
VM consolidation algorithm k
Period T
Control parameters

Fig. 3. Overview of the proposed approach

algorithms have internal, algorithm-specific parameters that influence their efficiency and effectiveness; for example, evolutionary algorithms can be tuned with parameters like population size, mutation rate etc. Similarly to the selection of the most appropriate algorithm for the given system state, also its most appropriate parameter configuration can be set on the fly, provided that the necessary rules are known. For example, in the case of heavy network traffic, the number of allowed migrations can be limited.

Beside selecting the algorithm and its parameters, a further customization possibility relates to the re-optimization period T, i.e., the time until the re-optimization cycle starts again. All previous works that we are aware of assumed T to be constant; however, this need not always be the case. If we choose a quick algorithm and limit it to just a few migrations so as to react quickly to an emergency situation, then it makes sense to lower T so that the next re-optimization happens earlier. This way, it can be checked in a timely manner whether the emergency has been resolved: if yes, other optimizations can be performed that were previously not done because of the higher-priority mitigation steps; if no, further measures can be taken to mitigate the issue.

5 Specific implementation

So far, we have described both the addressed problem and our proposed approach in a generic way. The reason is that the VM consolidation problem exists in many different flavors [22], but the presented approach can be applied to any variant in conceptually the same way. However, the specific algorithms that make up the portfolio, their parameters, as well as the specifics of the data center and the served workload may influence the details of how the proposed method should be applied. To validate our approach, we implemented it in a specific setting which we describe in the following.

5.1 Problem model

We focus on CPU usage as the most important resource for consolidation. The set of available PMs is denoted by P. Each PM $p \in P$ is associated with a CPU capacity c_p and power consumption w_p. The set of active or requested VMs is denoted by V. Each VM $v \in V$ is associated with a – current or predicted – CPU size s_v. The current mapping of VMs to PMs is given for a subset of the VMs $V_0 \subseteq V$ by $m_0 : V_0 \to P$. The aim is to determine a new mapping $m : V \to P$ of each VM to a PM that fulfills the capacity constraints, also leaving some headroom on each PM:

$$\forall p \in P : \sum_{v \in m^{-1}(p)} s_v \leq \lambda \cdot c_p. \tag{1}$$

Here, $m^{-1}(p)$ is the set of VMs mapped by m to PM p and $0 < \lambda \leq 1$ is a given constant, defining the headroom.

A further constraint is that the number of migrations should not be too high. The number of migrations can be computed as $|\{v \in V_0 : m(v) \neq m_0(v)\}|$.

The optimization objective is to minimize the total power consumption, which is given by the sum of the power consumption of the PMs that are active: $\sum\{w_p : p \in P_a\}$, where $P_a \subseteq P$ is the set of active PMs.

5.2 Used algorithms

We use a portfolio of two typical but very different algorithms. The first algorithm is the heuristic of Beloglazov et al. [5]. This is based on a packing heuristic called Modified Best Fit Decreasing (MBFD), in which the VMs to be placed are first sorted in non-increasing order of their CPU size, and then each VM is placed in the PM that can host it with the smallest increase in power consumption.

Newly requested VMs are placed directly using the MBFD heuristic. For re-optimizing the placement of existing VMs, the algorithm of Beloglazov et al. first determines the PMs whose utilization is above λ. From these PMs, some VMs are removed until their utilization gets below λ. The VMs removed this way are migrated to other PMs determined using again the MBFD heuristic. Finally, the algorithm tries for each PM whether it can be emptied by migrating all the VMs it hosts to some other PM – if this is possible, these migrations are carried out and the PM is shut down; otherwise, the migrations are not carried out.

The second algorithm consists of converting the VM consolidation problem to an integer linear program (ILP) and using an off-the-shelf ILP solver to solve it. The conversion mostly follows the approach of [3], and is described next.

Indexing VMs as v_i ($i = 1, \ldots, |V|$) and PMs as p_j ($j = 1, \ldots, |P|$), the following binary variables are introduced:

$$Alloc_{i,j} = \begin{cases} 1 & \text{if } v_i \text{ should be allocated on } p_j \\ 0 & \text{otherwise} \end{cases}$$

$$Active_j = \begin{cases} 1 & \text{if } p_j \text{ should be active} \\ 0 & \text{otherwise} \end{cases}$$

$$Migr_i = \begin{cases} 1 & \text{if } v_i \text{ should be migrated} \\ 0 & \text{otherwise} \end{cases}$$

Using these variables, the integer program can be formulated as follows ($i = 1, \ldots, |V|$ and $j = 1, \ldots, |P|$):

$$\min \quad \alpha \cdot \sum_{j=1}^{m} w_{p_j} \cdot Active_j + \mu \cdot \sum_{i=1}^{n} Migr_i \tag{2}$$

$$\text{s. t.} \quad \sum_{j=1}^{m} Alloc_{i,j} = 1 \qquad \forall i \tag{3}$$

$$Alloc_{i,j} \leq Active_j \qquad \forall i, j \tag{4}$$

$$\sum_{i=1}^{n} s_{v_i} \cdot Alloc_{i,j} \leq \lambda \cdot c_{p_j} \qquad \forall j \tag{5}$$

$$Migr_i = 1 - Alloc_{i,m_0(v_i)} \qquad \forall v_i \in V_0 \tag{6}$$

$$\sum_{i=1}^{n} Migr_i \leq K \tag{7}$$

$$Alloc_{i,j}, Active_j, Migr_i \in \{0, 1\} \qquad \forall i, j \tag{8}$$

The objective function (2) is the weighted sum of the total power consumption and the number of migrations ($\alpha, \mu \geq 0$ are given weights). Equation (3) ensures that each VM is allocated to exactly one PM, whereas constraint (4) ensures that for a PM p_j to which at least one VM is allocated, $Active_j = 1$. Together with the objective function, this ensures that $Active_j = 1$ holds for *exactly* those PMs that accommodate at least one VM. Constraint (5) is the capacity constraint. Equation (6) determines the values of the $Migr_i$ variables and equation (7) constrains the number of migrations ($K > 0$ is a given constant).

5.3 Algorithm and parameter selection logic

Our selection logic is based on a simple but powerful indicator of the current system state: the number of PMs currently not satisfying Equation (1). If this number, denoted as L, is higher than a predefined threshold L_0, then we assume that a quick reaction is necessary; otherwise, the reaction can be more relaxed.

The rationale behind using this metric is the following. We can assume that in the previous re-optimization cycle, VMs were re-distributed among PMs in

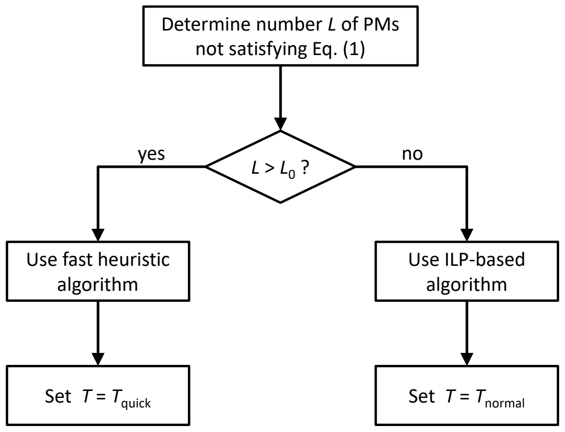
Determine number L of PMs not satisfying Eq. (1)
$L > L_0$?
yes
no
Use fast heuristic algorithm
Use ILP-based algorithm
Set $T = T_{\text{quick}}$
Set $T = T_{\text{normal}}$

Fig. 4. Specific algorithm and parameter selection strategy

such a way that the utilization of each PM is below λ, and for most PMs it is near λ. If the workload is in an upturn, then the PMs whose load was just under the limit will exceed the limit; and indeed a quick reaction is needed to avoid negative consequences of further load increase. On the other hand, if the workload is stagnating or decreasing, then the load of the PMs still satisfies Equation (1). In this case, there is more time to determine the new placement of the VMs. Hence the number of PMs not satisfying Equation (1) is indeed a good indicator of how quickly a reaction is needed.

We assume that the heuristic of Beloglazov et al. is significantly faster than the ILP-based algorithm, but typically the ILP-based algorithm delivers better results. For this reason, we apply the fast heuristic if a quick reaction is necessary (i.e., $L > L_0$) and the ILP-based algorithm otherwise (see also Fig. 4).

Also the time T until the next re-optimization cycle is set adaptively, based on a similar decision logic. If the ILP-based algorithm is carried out, then T is set to its normal value. However, if we established that a quick reaction is necessary and hence run the heuristic algorithm, then we set T to a lower value. The reason is that we should keep the ability to respond quickly if the workload continues to rise. Cloud workloads are known to be amenable to the flash crowd phenomenon, which can quickly lead to severe violation of service level objectives. This is why we have to be careful if the load starts to rise. On the other hand, if the load is not rising, performing VM consolidation too often would be counterproductive because of the overhead associated with migrations.

6 Empirical results

Simulations were used to assess the effects of our adaptive VM consolidation approach, using the CloudSim simulator [6], version 4.0. CloudSim already contains the VM consolidation algorithm of Beloglazov et al. We implemented the

Table 1. Aggregated results of the experiments

Workload	Energy [kWh]			Overloads		
	heuristic	ILP	portfolio	heuristic	ILP	portfolio
constant	11.63	9.04	9.04	0	0	0
decrease	16.60	15.21	14.93	0	0	0
increase	18.90	19.11	20.12	146	60	39
peak	18.99	16.09	18.51	104	60	2
valley	16.15	14.81	15.93	133	135	40
sinus small	12.39	10.35	9.88	0	0	0
sinus big	18.77	17.40	17.15	75	69	13
total	113.43	102.01	105.56	458	324	94

ILP-based algorithm using the Gurobi Optimizer, version 7.0.2. In addition, we implemented the algorithm and parameter selection logic described in Section 5.3. In all cases, λ was set to 0.8.

We simulate a cluster of 100 PMs serving 500 VMs. The PMs belong to three types (with one third of the PMs belonging to each type), having CPU capacities of 2000, 4000, and 8000 MIPS. The VMs' requested CPU size ranges from 200 to 1500 MIPS, and their actual CPU size is always defined as percentage of their requested size, as explained below. Re-optimization is normally carried out every 5 minutes (i.e., $T_{normal} = 300s$) like in many previous works (e.g., [21]). When T should be reduced to respond quickly, it is set to $T_{quick} = T_{normal}/2 = 150s$. The ILP-based algorithm is given a time budget of 60 seconds; the execution time of the heuristic algorithm is negligible (it was below 1 second in all of our experiments). Migrations take on average about 32 seconds. The power consumption of a running PM is 400W. The experiments were performed on a Lenovo ThinkPad X1 laptop with Intel Core i5-4210U CPU @ 1.70GHz and 8GB RAM.

We tested several different workload patterns to assess how our approach works in different settings. Each pattern takes 1 hour. In each case, the proposed approach is compared with the two pure strategies of using always the ILP-based algorithm or always the heuristic algorithm. The criteria for comparison are the number of active PMs, the total energy consumption, and the number of times a PM was overloaded, where the latter is assessed every 60 seconds.

The results of the experiments are summarized in Table 1. In the first experiment, the workload was constant 50% of the requested capacity. As expected, all algorithms were able to perform consolidation without incurring PM overloads. The ILP-based algorithm resulted in about 22% reduction in power consumption compared to the heuristic algorithm. Since there was no rise in the workload, the portfolio-based approach always chose the ILP-based algorithm, hence it led to the same result.

The situation is similar in the second experiment, in which the load decreases from 90% to 10% of the requested capacity. Again, there was no PM overload. The three algorithms led to similar energy consumption, with the portfolio-

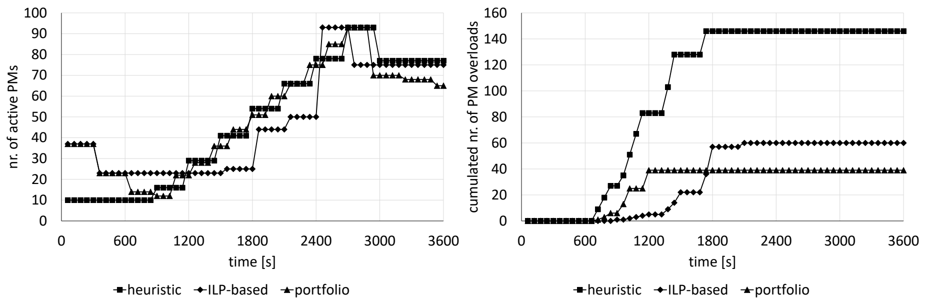

Fig. 5. Effects of the load increasing from 10% to 90%

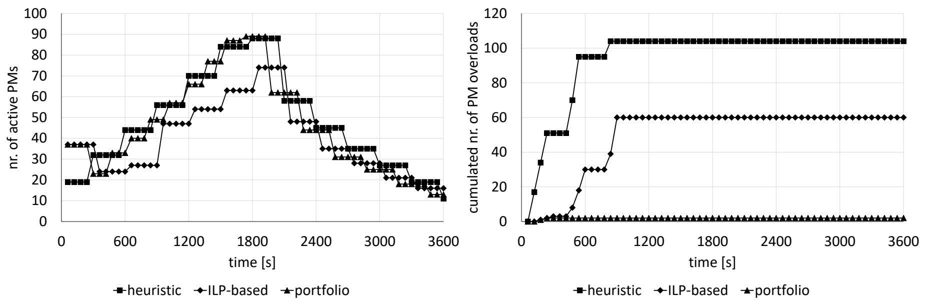

Fig. 6. Effects of the load first increasing from 10% to 90%, then decreasing back to 10%

based approach leading to about 10% reduction in energy consumption over the heuristic and about 2% over the ILP-based algorithm[2].

In the third experiment, the opposite happens: the load is increased from 10% to 90%. As can be seen, the order of the algorithms also becomes opposite: now the portfolio-based approach leads to 5-6% higher energy consumption than the others. However, there are considerable differences in terms of PM overloads: the ILP-based algorithm leads to 54% more PM overloads, the heuristic to 274% more PM overloads than the portfolio-based approach. The details are shown in Fig. 5. As can be seen, the portfolio-based approach can react faster to the change than the other algorithms.

The fourth experiment, called "peak," is a combination of the preceding two: in the first half of the time window, the load increases from 10% to 90%, then in the second half it decreases back to 10%. As can be seen, the energy consumption achieved by the three evaluated approaches is again very similar to each other; however, in terms of the number of PM overloads, the portfolio-based approach is again clearly superior. The details are shown in Fig. 6. Not surprisingly, the

[2] In almost all re-optimization cycles, the portfolio-based approach chooses the ILP-based algorithm, hence they behave almost the same. The only exception is the first cycle: since the workload starts at 90%, the initial placement of the VMs leads to several PMs with utilization above λ, so that the fast heuristic is chosen.

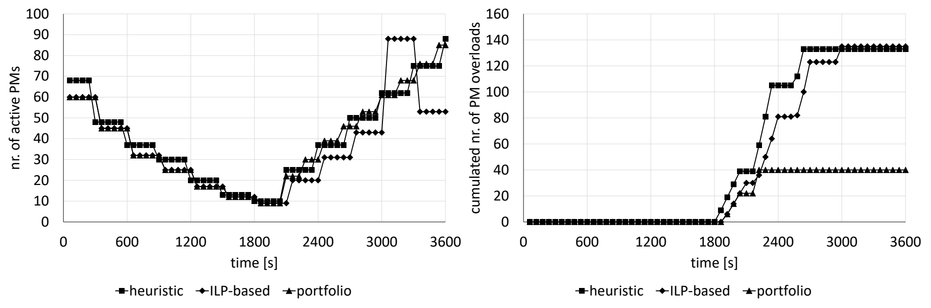

Fig. 7. Effects of the load first decreasing from 90% to 10%, then increasing back to 90%

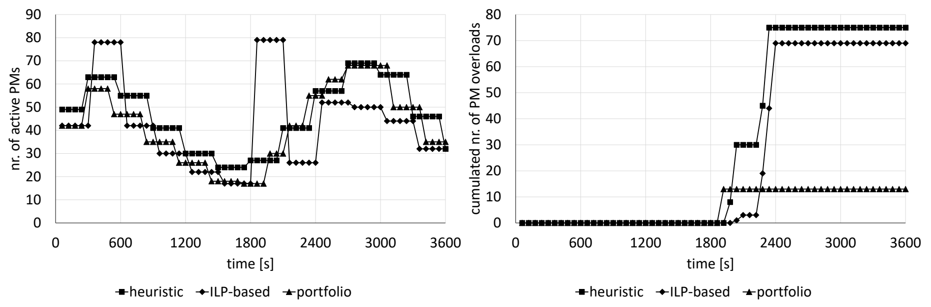

Fig. 8. Effects of the load following a sinus curve around 50% with amplitude 20%

difference between the three approaches arises in the first half of the time window, where the portfolio-based approach provides faster and better reaction to the change than the others.

The fifth experiment, called “valley,” is the opposite of the previous one: in the first half of the time window, the load decreases from 90% to 10%, then in the second half it increases back to 90%. The results, shown in detail in Fig. 7, are similar to the previous ones.

In the next two experiments, the load follows a sinus curve around 50%. In the experiment termed “sinus small,” the amplitude of the sinus curve is 5%, whereas in the “sinus big” experiment it is 20%. In both cases, the portfolio-based approach outperforms the other two. The details for the “sinus big” experiment are shown in Fig. 8.

The aggregated figures (last line of Table 1) reveal that the portfolio-based approach improves energy consumption by 7% compared to the heuristic, but this result is still 3% worse than that of the ILP-based algorithm. Concerning the number of overloads, the portfolio-based approach emerges as clear winner.

7 Conclusions and future work

In this paper, we investigated how different algorithms for the VM consolidation problem can be combined into an algorithm portfolio from which an automated decision-making mechanism can choose dynamically at run-time based on the current system state. This way, the complementary advantages of different algorithms can be leveraged. In particular, we have shown how a fast but simple heuristic can be combined with a more sophisticated but slow ILP-based algorithm. Beside the choice of algorithm, also algorithm parameters as well as the re-optimization interval can be chosen by the same mechanism.

The simulation results demonstrate that the suggested approach is promising because in most cases it leads to a better trade-off between energy consumption and PM overloads than the two underlying algorithms.

Obviously, our current implementation is rather simple and could be improved in several ways. For example, the used knowledge about the two underlying algorithms is simplistic: the ILP-based algorithm is assumed to always lead to better quality than the heuristic. In reality, this is not always the case, so that a more sophisticated model of the algorithms' performance could result in better decisions. The model of algorithm performance could also be learned during run-time through appropriate machine learning techniques.

Also the decision-making is based on a simple metric. More intelligence could be added, for instance in the form of time series analysis, to make better decisions. Further possibilities include the addition of more algorithms to the portfolio or running multiple algorithms from the portfolio in parallel if sufficient parallel resources are available.

Acknowledgments

This work was partially supported by the Hungarian Scientific Research Fund (Grant Nr. OTKA 108947) and by the European Union's Horizon 2020 research and innovation programme under grant 731678 (RestAssured).

References

1. Ábrahám, E., Corzilius, F., Johnsen, E.B., Kremer, G., Mauro, J.: Zephyrus2: On the fly deployment optimization using SMT and CP technologies. In: Proceedings of the 2nd International Symposium on Dependable Software Engineering. pp. 229–245 (2016)
2. Ahvar, E., Ahvar, S., Mann, Z.A., Crespi, N., Garcia-Alfaro, J., Glitho, R.: CACEV: a cost and carbon emission-efficient virtual machine placement method for green distributed clouds. In: Proceedings of the 13th IEEE International Conference on Services Computing. pp. 275–282 (2016)
3. Bartók, D., Mann, Z.A.: A branch-and-bound approach to virtual machine placement. In: Proceedings of the 3rd HPI Cloud Symposium "Operating the Cloud". pp. 49–63 (2015)

4. Beloglazov, A., Abawajy, J., Buyya, R.: Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Generation Computer Systems 28, 755–768 (2012)
5. 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 24(13), 1397–1420 (2012)
6. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience 41(1), 23–50 (2011)
7. Cosmo, R.D., Lienhardt, M., Treinen, R., Zacchiroli, S., Zwolakowski, J., Eiche, A., Agahi, A.: Automated synthesis and deployment of cloud applications. In: ACM/IEEE International Conference on Automated Software Engineering. pp. 211–222 (2014)
8. Digital Power Group: The cloud begins with coal – Big data, big networks, big infrastructure, and big power (2013)
9. Gao, Y., Guan, H., Qi, Z., Hou, Y., Liu, L.: A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. Journal of Computer and System Sciences 79, 1230–1242 (2013)
10. García-Valls, M., Cucinotta, T., Lu, C.: Challenges in real-time virtualization and predictable cloud computing. Journal of Systems Architecture 60(9), 726–740 (2014)
11. Gmach, D., Rolia, J., Cherkasova, L., Belrose, G., Turicchi, T., Kemper, A.: An integrated approach to resource pool management: Policies, efficiency and quality metrics. In: IEEE International Conference on Dependable Systems and Networks. pp. 326–335 (2008)
12. Gomes, C.P., Selman, B.: Algorithm portfolios. Artificial Intelligence 126(1-2), 43–62 (2001)
13. Guazzone, M., Anglano, C., Canonico, M.: Exploiting VM migration for the automated power and performance management of green cloud computing systems. In: 1st International Workshop on Energy Efficient Data Centers. pp. 81–92. Springer (2012)
14. Guenter, B., Jain, N., Williams, C.: Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning. In: Proceedings of IEEE INFOCOM. pp. 1332–1340. IEEE (2011)
15. Huberman, B.A., Lukose, R.M., Hogg, T.: An economics approach to hard computational problems. Science 275(5296), 51–54 (1997)
16. Hyser, C., McKee, B., Gardner, R., Watson, B.J.: Autonomic virtual machine placement in the data center. Tech. rep., HP Laboratories (2008)
17. Jung, G., Hiltunen, M.A., Joshi, K.R., Schlichting, R.D., Pu, C.: Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. In: IEEE 30th International Conference on Distributed Computing Systems. pp. 62–73 (2010)
18. Li, H., Zhu, G., Cui, C., Tang, H., Dou, Y., He, C.: Energy-efficient migration and consolidation algorithm of virtual machines in data centers for cloud computing. Computing 98(3), 303–317 (2016)
19. Li, R., Zheng, Q., Li, X., Wu, J.: A novel multi-objective optimization scheme for rebalancing virtual machine placement. In: IEEE 9th International Conference on Cloud Computing. pp. 710–717 (2016)

20. Li, W., Tordsson, J., Elmroth, E.: Virtual machine placement for predictable and time-constrained peak loads. In: Proceedings of the 8th International Conference on Economics of Grids, Clouds, Systems, and Services (GECON 2011). pp. 120–134. Springer (2011)
21. Li, Z., Yan, C., Yu, X., Yu, N.: Bayesian network-based virtual machines consolidation method. Future Generation Computer Systems 69, 75–87 (2017)
22. Mann, Z.A.: Allocation of virtual machines in cloud data centers – a survey of problem models and optimization algorithms. ACM Computing Surveys 48(1) (2015)
23. Mann, Z.A.: Approximability of virtual machine allocation: much harder than bin packing. In: Proceedings of the 9th Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications. pp. 21–30 (2015)
24. Mann, Z.A.: Modeling the virtual machine allocation problem. In: Proceedings of the International Conference on Mathematical Methods, Mathematical Models and Simulation in Science and Engineering. pp. 102–106 (2015)
25. Mann, Z.A.: Multicore-aware virtual machine placement in cloud data centers. IEEE Transactions on Computers 65(11), 3357–3369 (2016)
26. Mann, Z.Á., Szabó, M.: Which is the best algorithm for virtual machine placement optimization? Concurrency and Computation: Practice and Experience 29(10) (2017)
27. Marotta, A., Avallone, S.: A simulated annealing based approach for power efficient virtual machines consolidation. In: Proceedings of the 8th IEEE International Conference on Cloud Computing. pp. 445–452 (2015)
28. Mishra, M., Sahoo, A.: On theory of VM placement: Anomalies in existing methodologies and their mitigation using a novel vector based approach. In: IEEE International Conference on Cloud Computing. pp. 275–282 (2011)
29. Qu, C., Calheiros, R.N., Buyya, R.: Mitigating impact of short-term overload on multi-cloud web applications through geographical load balancing. Concurrency and Computation: Practice and Experience (2017)
30. Rampersaud, S., Grosu, D.: Sharing-aware online algorithms for virtual machine packing in cloud environments. In: Proceedings of the 8th IEEE International Conference on Cloud Computing. pp. 718–725 (2015)
31. Ribas, B.C., Suguimoto, R.M., Montano, R.A.N.R., Silva, F., de Bona, L., Castilho, M.A.: On modelling virtual machine consolidation to pseudo-Boolean constraints. In: 13th Ibero-American Conference on AI. pp. 361–370 (2012)
32. Salehi, M.A., Krishna, P.R., Deepak, K.S., Buyya, R.: Preemption-aware energy management in virtualized data centers. In: 5th International Conference on Cloud Computing. pp. 844–851. IEEE (2012)
33. Shi, L., Furlong, J., Wang, R.: Empirical evaluation of vector bin packing algorithms for energy efficient data centers. In: IEEE Symposium on Computers and Communications. pp. 9–15 (2013)
34. Strunk, A.: Costs of virtual machine live migration: A survey. In: 8th IEEE World Congress on Services. pp. 323–329 (2012)
35. Svärd, P., Li, W., Wadbro, E., Tordsson, J., Elmroth, E.: Continuous datacenter consolidation. In: IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom). pp. 387–396 (2015)
36. Tomás, L., Tordsson, J.: An autonomic approach to risk-aware data center overbooking. IEEE Transactions on Cloud Computing 2(3), 292–305 (2014)
37. Verma, A., Dasgupta, G., Nayak, T.K., De, P., Kothari, R.: Server workload analysis for power minimization using consolidation. In: Proceedings of the 2009 USENIX Annual Technical Conference. pp. 355–368 (2009)

38. Wood, T., Shenoy, P., Venkataramani, A., Yousif, M.: Sandpiper: Black-box and gray-box resource management for virtual machines. Computer Networks 53(17), 2923–2938 (2009)
39. Xiao, Z., Song, W., Chen, Q.: Dynamic resource allocation using virtual machines for cloud computing environment. IEEE Transactions on Parallel and Distributed Systems 24(6), 1107–1117 (2013)
40. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: SATzilla: portfolio-based algorithm selection for SAT. Journal of Artificial Intelligence Research 32, 565–606 (2008)
41. Zhang, Z., Hsu, C.C., Chang, M.: CoolCloud: A practical dynamic virtual machine placement framework for energy aware data centers. In: Proceedings of the 8th IEEE International Conference on Cloud Computing. pp. 758–765 (2015)
42. Zheng, Q., Li, R., Li, X., Wu, J.: A multi-objective biogeography-based optimization for virtual machine placement. In: Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. pp. 687–696 (2015)