Mosa and Paton *Journal of Cloud Computing: Advances, Systems and Applications* (2016) 5:17
DOI 10.1186/s13677-016-0067-7

Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH Open Access

Optimizing virtual machine placement for energy and SLA in clouds using utility functions

Abdelkhalik Mosa* and Norman W. Paton

Abstract

Cloud computing provides on-demand access to a shared pool of computing resources, which enables organizations to outsource their IT infrastructure. Cloud providers are building data centers to handle the continuous increase in cloud users' demands. Consequently, these cloud data centers consume, and have the potential to waste, substantial amounts of energy. This energy consumption increases the operational cost and the CO_2 emissions. The goal of this paper is to develop an optimized energy and SLA-aware virtual machine (VM) placement strategy that dynamically assigns VMs to Physical Machines (PMs) in cloud data centers. This placement strategy co-optimizes energy consumption and service level agreement (SLA) violations. The proposed solution adopts *utility functions* to formulate the VM placement problem. A genetic algorithm searches the possible VMs-to-PMs assignments with a view to finding an assignment that maximizes utility. Simulation results using CloudSim show that the proposed utility-based approach reduced the average energy consumption by approximately 6 % and the overall SLA violations by more than 38 %, using fewer VM migrations and PM shutdowns, compared to a well-known heuristics-based approach.

Keywords: Cloud computing, Virtual machine placement, Cloud resource management, Utility functions, Energy-aware, Service level agreement (SLA)

Introduction

Cloud computing delivers application, platform or infrastructure services to large numbers of users with diverse and dynamically changing requirements. To meet the expectations of their users in a cost-effective manner, cloud service providers must make numerous resource management decisions that satisfy different objectives, for example to meet Service Level Agreements (SLAs) while minimizing energy costs [1].

In this paper, we focus on the problem of *adaptively allocating virtual machines (VMs) to physical hosts*, in the context of unpredictable workloads. Specifically, this involves making decisions such as *when to relocate VMs*, *which VMs to relocate*, *where to place VMs that are to be relocated*, and *which physical machines can be switched off*. These decisions can be made with a view to meeting different objectives; in our case, we seek to maximize the profit of an Infrastructure-as-a-Service (IaaS) service provider by trading off income (which involves meeting SLAs) and expenditure (which involves saving energy by moving physical machines that are not needed into power saving modes).

We are not the first to address this problem. For example, in a series of papers Beloglazov et al. [2–4] develop heuristic algorithms that make dynamic workload allocation decisions, taking into account energy usage when deciding where to place VMs. In adopting a heuristic approach, these papers focus on identifying criteria that suggest that an adaptation may be beneficial (which tends to involve detecting which hosts are over- or under-loaded), and then making reallocation decisions in ways that take into account estimated energy usage. In developing and refining their heuristics, the authors focus on challenges like detecting when the load on a physical node suggests that an adaptation may be beneficial. By systematically refining their heuristics, the authors were able to make proposals that significantly improved on their static counterparts.

We note, however, that such heuristic approaches do not address the objective of the problem directly. Here, the goal is to meet SLAs while conserving energy, but the focus of the heuristics, that determine when adaptations should take place and which VMs should be moved, is on

*Correspondence: amosa@cs.man.ac.uk
School of Computer Science, University of Manchester, Oxford Road, M13 9PL Manchester, UK

the load on physical nodes. Clearly, the load on physical nodes is relevant to this problem, but it is effectively a proxy for the goal. In this paper, we adopt what is referred to as a utility-based approach to adaptive energy-aware virtual machine placement. In the utility-based approach to adaptive systems [5–7], a *utility function* is defined that specifies the goal of the adaptation, and an optimization algorithm explores alternative adaptations, to identify the adaptation that maximizes utility. In this paper, the utility of an assignment a over a time interval t is defined as: $Utility(a,t) = Income(a,t) - EnergyCost(a,t)$.

This utility function captures the objective for the service provider. The *Utility(a,t)* returns the predicted financial return over a period of time t of an assignment a of virtual machines (VMs) to physical machines (PMs). To apply this in practice involves the development of models for estimating the *Income* and *EnergyCost* of an assignment over a time interval *t*, and the selection of a search function that explores the space of alternative assignments. Thus, the utility based approach captures the goal of the adaptation explicitly and searches for solutions that meet the goal. The utility based approach has been applied to a range of applications, from workflow scheduling on grids [8] to data center cooling [9], and here is applied to adaptive VM placement.

The key contributions of this paper are summarized as follows:

1. An application of the utility-based approach to the VM placement problem.
2. A utility function that estimates the profit from adaptive VM placement taking into account the impact of adaptations, the energy used, and the SLA violations.
3. An empirical evaluation of the proposed solution using the CloudSim simulation framework.

The paper is structured as follows. In "Related work" section, we describe related work on autonomic virtual machine placement and utility-based resource allocation. In "Optimizing virtual machine placement" section, we provide a precise description of the VM placement problem. "Utility-based resource allocation" section indicates how the utility-based approach has been applied to the VM placement problem, and "Experimental evaluation" section empirically evaluates the approach, comparing it to an existing heuristic strategy. Conclusions and future directions are discussed in "Conclusions and further work" section.

Related work

This section discusses work that is related to that described in the paper, considering in turn *virtual machine placement* and *utility based resource allocation*. We do not re-review less closely related research in cloud computing, of which there is a considerable amount, and for which review articles already exist (e.g. [1, 10, 11]).

Virtual machine placement: In relation to *virtual machine placement*, we review results in terms of *when* placement decisions are made, the *objective* that placement decisions seek to meet, and the *decision-making paradigm* that is used to identify a suitable allocation.

In terms of *when* placement decisions are made, approaches can be considered to be *static* (e.g. [12–14]), in which decisions once made are not reviewed during the lifetime of the VM, or *dynamic* (e.g. [2, 15–18]), in which the VM to PM assignment may change during the execution of the VM. As *dynamic* approaches often make use of information on the actual load that is not available to static approaches, dynamic approaches often use a static approach for initial placement. In the remainder of this section, we will focus on dynamic approaches, as these are the most relevant to this paper.

Virtual machine placement decisions can involve trade-offs, for example between energy usage and risk of SLA violations, so methods implicitly or explicitly seek to meet an *objective*. Dynamic VM placement techniques have been developed that seek to maximize revenue (e.g. [12]), conserve energy (e.g. [17, 19, 20]), and to meet SLAs while saving energy where possible (e.g. [2, 21, 22]). In this paper, we seek to maximize profit, by making allocation decisions that take into account the cost of energy usage and the loss of income that would result from the violation of SLAs.

Researchers have investigated a range of *decision-making paradigms*, which decide when to change the VM to PM assignment, what migrations to carry out, and perhaps also what PMs can be turned off to save energy. Dynamic VM placement proposals have employed heuristics (e.g. [2, 12, 16, 17]), integer linear programming (e.g. [12]) and bespoke algorithms (e.g. [15]). In this paper, we deploy a utility based approach, in which an evolutionary search is used to explore alternative allocations.

We know of no other work that has both addressed the same problem as we address (*dynamic* virtual machine placement with the *objective* of maximizing profit, taking into account both SLAs and energy) using a utility-based *decision-making paradigm*. However, to enable the evaluation of our approach, we compare our results with a proposal that addresses the same problem, but using a different *decision-making paradigm*. Beloglazov et al. [2, 3, 23] proposed a heuristics-based energy-aware resource management system that meets quality of service (QoS) requirements. Their solution follows the divide and conquer concept by dividing the main problem into 4 sub-problems namely, *host overload detection*,

host underload detection, *VM selection* and *VM placement*. The *host overload detection* problem determines when a host is considered to be overloaded. Once the host is considered overloaded, VMs are selected for migration from this overloaded host to a non-overloaded one. The *host underload detection* problem involves deciding that a host is underloaded. After host underload detection, all VMs in the underloaded host should be migrated to another host if possible, thereby enabling that host to be placed in a power saving mode. The *VM selection* problem involves deciding which VMs should be selected for migration from the overloaded host. Finally, the *VM placement* problem involves choosing the appropriate host for migrating the VMs to. Further details of this approach are provided in "Experimental evaluation" section, where it is compared with our utility-based proposal.

Utility-based resource allocation: In this paper, we use utility functions to compare alternative adaptations, and thereby to select the adaptation that is expected to yield the highest utility. As discussed by Kephart et al. [6], decision-making in computing involves a transition from a current state into one of several alternative future states, by way of candidate adaptations. In this context, it is the role of the utility function to quantify the suitability of each of the alternative future states. In the original proposal for utility-based adaptation [5], a *controller* is responsible for selecting the collection of *control parameters* that maximize the utility function, using a *utility calculator* that implements a model of the environment. Utility functions have been applied in autonomic computing for: configuring the properties of application hosting environments such as web servers [5], for selecting between alternative providers of a service [24], for managing the physical environment within data centres [9], for allocating jobs within a collection of workflows to machines on a grid [8], for optimizing resource utilization for scientific applications [25], and for balancing the load of a collection of database queries over the nodes in a cluster [26]. As the designs of these applications have various features in common, a methodology has been proposed for the development of utility-based applications [27], which we follow in "Utility-based resource allocation" section.

Optimizing virtual machine placement

This section demonstrates the virtual machine placement problem by describing the input, processing, and output model in "Input, processing and output for the VM placement" section. Moreover, "Monitoring, analysis, planning and execution of VM placement" section describes monitoring information that is used for the analysis, planning and the actual execution of the VM placement.

Input, processing and output for the VM placement

Figure 1 illustrates the input, processing, and output of the VM placement problem. The summary of the problem is as follows:

- Input: Given a cloud data center with N heterogeneous physical machines (PMs) with limited resource capabilities, the cloud provider receives VM placement requests consisting of M virtual machines (VMs) which need to be assigned to the PMs. Finally, the cloud user runs application workloads on the VMs. In the conducted experiments, the VMs come in batch mode, however they can be online.

- Processing: The processing of the VM placement problem involves identifying an assignment a that associates each $vm_i \in VM$, with a single $pm_j \in PM$. The processing consists of the *initial placement* controller and *dynamic placement* controller.

  - The *initial placement* controller either accepts or rejects VM placement requests based on the resource constraints in the data center. This is simply a bin packing problem and there are dozens of techniques for solving it in the literature.
  - The *dynamic placement* controller periodically reallocates existing VMs based on resource utilization. The goal of the dynamic placement is to save energy while minimizing SLA violations (SLAVs).

- Output: A VMs-to-PMs assignment.

In this paper, we develop a *utility based* solution for the *dynamic placement* problem. Furthermore, we evaluate the proposed solution and compare it with a *heuristics-based* one proposed in [2, 3]. The dynamic placement based on VM consolidation is a strictly NP-hard problem [28].

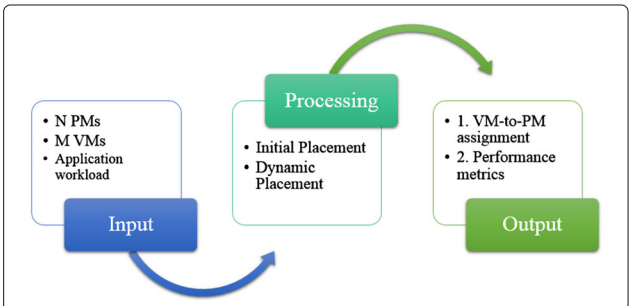

Fig. 1 IPO model. Input, processing and output for the VM placement

Monitoring, analysis, planning and execution of VM placement

Figure 2 shows the loop design model of the proposed utility-based solution in terms of monitoring, analysis, planning and execution (MAPE). The *monitoring (M)* component monitors the CPU utilization of the PMs and the VMs based on the applications' workload in the cloud data center. By the end of each scheduling interval, the *analysis (A)* component searches for alternative VMs-to-PMs assignments based on the collected monitoring data. The search process runs periodically depending on the scheduling interval. The *planning (P)* performs a utility-based assessment of candidate VMs-to-PMs assignments and considers the best one found so far. Moreover, it finds which VMs are going to be migrated, if any, and to where, based on the best found VMs-to-PMs assignment. Finally, the *execution (E)* component performs the migration of the VMs in addition to switching the unused PMs off.

Utility-based resource allocation

Abstracting from several experiences of using the utility based approach, a methodology has been developed for its application [7], which we follow here for applying the approach to VM placement.

Utility property selection

Selecting the properties of the utility function is crucial as they affect the value of the utility function. From [7], utility property selection involves selecting the property that it would be desirable to maximize. Our solution tries to maximize the profit, which is derived from *income*, *energy cost* and *violation costs*. The income represents the money that the cloud provider receives from cloud customers due to hosting their VMs. Energy cost represents the cost of the energy consumed due to placing the VMs and the running of applications. Finally, the violation costs represent the amount of money that the cloud provider has to pay to cloud customers due to SLA violations (SLAVs).

Utility function definition

The utility function expresses the self-managing policy adopted for creating an autonomic cloud data center. The goal of the utility function is to maximize the profit of VM placement by minimizing energy consumption and SLA violations (SLAVs). Equation (1) provides the high-level definition of the utility function.

$$\begin{aligned} Utility(a,t) = Income(a,t) - (&EstimatedEnergyCost(a,t) \\ &+ EstimatedViolationCost(a,t) + PDMCost(a,t)) \end{aligned} \tag{1}$$

Here, a is a map representing the VMs-to-PMs assignment, and t is the assignment time period. $Income(a,t)$ is the total income from hosting the customers' VMs; $EstimatedEnergyCost(a,t)$ is the expected cost of energy consumption due to the assignment. *EstimatedViolationCost(a,t)* represents the cost of SLA violation due to the over-utilization of the hosting PMs, a cost that is calculated based on the number of VMs in violation. VMs are not available during migration and hence there should be a way to estimate the cost resulting from this violation. *PDMCost(a,t)* represents the violation cost of the *performance degradation due to the migration* (PDM) of VMs among PMs. The utility function requires the development of models for estimating both the *EnergyCost* and the *violation costs* of an assignment over a time interval t, which are provided in "Utility model development" section. This is followed by a description of a genetic algorithm that searches the space of VM-to-PM assignments for solutions that maximize the utility.

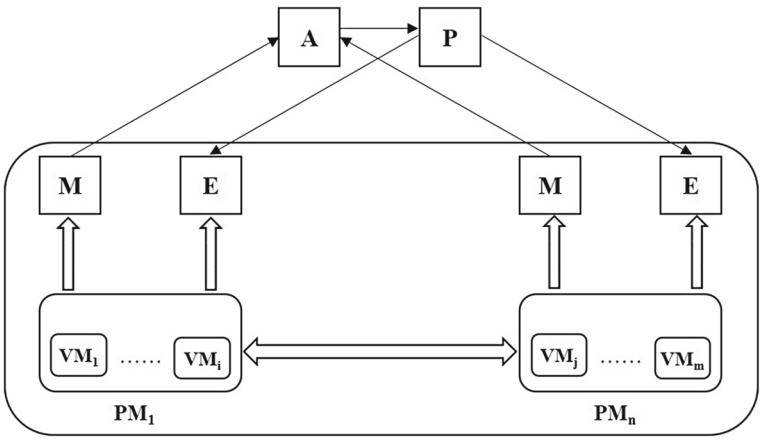

Fig. 2 MAPE of VM placement. Monitoring, Analysis, Planning and Execution (MAPE) loop design model of the VM placement problem

Utility model development

The utility model estimates the *expected energy cost* and the two types of *SLA violations* due to the VMs assignment. The utility model calculations depend on the percentage of *CPU utilization*, which is calculated as shown in the following sub-section.

Calculating PM's CPU utilization based on VM utilization

Algorithm 1 estimates the CPU utilization of the PM based on the candidate assignments. Algorithm 1 uses the captured monitoring information to calculate the CPU utilization of each of the PMs in the candidate assignment. Algorithm 1 depends on two assignments, the *current assignment* and the *candidate assignment*. The current assignment (c), is the currently running VMs-to-PMs assignment. The genetic algorithm produces a number of candidate assignments that need to be compared to the current one. The candidate assignment (a), represents an assignment that might or might not be adopted. If the best candidate assignment is better than (i.e. has a higher fitness value than) the current assignment then the candidate assignment should replace the current one. A new assignment means that some VMs might be removed from a PM while others might be added according to the new VMs-to-PMs mapping. The key idea is passing the candidate assignment a as a parameter to the *EstimatedUtilization(a)* function for computing the expected utilization due to the assignment.

Algorithm 1: The expected CPU utilization of each PM

```
allPmsUtilizations[] EstimatedUtilization(a)
Result: The CPU utilization of each PM in the
        cloud data center due to the candidate
        assignment
Initialization:
assignment c ← getCurrentAssignment(pm);
addedVms ← VMs associated with the PM in a
that are not in c;
removedVms ← VMs associated with the PM in c
that are not in a;
allPmsUtilizations[] ← 0;
u[pm] ← 0;
foreach pm in the Assignment a do
    u[pm] ←pm.getUtilization()+
           sum(addedVms.getUtilization())−
           sum(removedVms.getUtilization());
    allPmsUtilizations[ ].add(u[pm] );
end
return allPmsUtilizations[];
```

In Algorithm 1, *pm* refers to the physical machine; *pm.getUtilization*() returns *pm*'s current CPU utilization in millions of instructions per second (MIPS) and *added.getUtilization*() returns the total CPU utilization of VMs that are in the candidate, a, assignment and not in the current, c, one. The *removed.getUtilization*() call returns the total CPU utilization of VMs that are in the current assignment c and not in the candidate, a, one; and $u[pm]$ stores the utilization of each PM in MIPS. Finally, *allPmsUtilizations*[ ] is an array holding the utilization of each single PM in the cloud data center.

Energy cost estimation

Algorithm 2 calculates the expected cost of energy due to the candidate assignment during the time interval t. For computing power consumption, a power model should be used. In this work, real power consumption data is used for calculating the power consumption. These real consumption data are provided in the *SPECpower_ssj*2008 [3] benchmark. These data are provided with the CloudSim simulator through two different servers, *HP ProLiant ML110 G4* and *HP ProLiant ML110 G5*. Figure 3, from [3], exhibits a table that shows the variation of power consumption of those two servers according to the level of utilization in Watts.

Algorithm 2: Energy Cost Estimation

```
float EstimatedEnergyCost(a, t)
Result: Total Energy cost of the running PMs
Initialization:
powerConsumedPerPm[] = 0;
totalPowerConsumed = 0;
totalEnergyCost = 0;
allPmsUtilizations[] = EstimatedUtilization(a);
foreach pm in the Assignment a do
    powerConsumedPerPm[pm] =
    pm.getPowerModel().getPower
    (cpuUtilization[pm]);
    totalPowerConsumed +=
    powerConsumedPerPm[pm];
end
totalEnergyCost = totalPowerConsumed *
energyCostPerSec * t;
return(totalEnergyCost);
```

The function *getPower(cpuUtilization[pm])* returns the power according to the power model used and *EstimatedUtilization*(a) is shown in Algorithm 1. The return value from this algorithm is the cost of the total energy consumed due to the assignment in the cloud data center in the specified time period t.

Server	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
HP ProLiant G4	86	89.4	92.6	96	99.5	102	106	108	112	114	117
HP ProLiant G5	93.7	97	101	105	110	116	121	125	129	133	135

Fig. 3 Power consumption at different utilizations, from [3]. Power consumption for *HP ProLiant G4* and *G5* provided by SPECpower benchmark

Violation cost due to PM over-utilization

Many factors might lead to SLA violations in a virtualized environment such as the over-utilization of the PMs, the migration of VMs, performance interference among co-located workloads [29], overheads from co-locating network-intensive or CPU-intensive workloads in isolated VMs, and system failure or outage. However, we only consider the over-utilization of the PMs and the VMs migration, as shown in "Violation cost due to VM migrations" section, for comparison purposes, and as there is no interference between the VMs that constitute the workload used in the experiments in "Experimental evaluation" section. Algorithm 3 calculates the violation cost resulting from the over-utilization of the PMs based on the number of VMs in violation.

VMList represents the list of VMs assigned to each PM, and *demand* represents the total CPU demand of the PM resulting from the currently assigned VMS. The *sort*(*VMList*) method sorts the VMs in descending order according to the CPU demand. Descending order was used so as to minimize the number of VMs in violation. If the CPU *demand* is greater than the actual PM's CPU capabilities (*supply*), this means that there is a violation and the number of VMs' in violation will be counted.

Algorithm 3: Violation cost due to PMs' over-utilization

```
float EstimatedViolationCost(a, t)
Result: SLA violation cost due to PMs'
        over-utilization
Initialization:
numOfVMsInViolation ← 0; violation ← 0;
foreach pm in the Assignment a do
    VMList ← list of VMs from a in pm ;
    VMList ← sort(VMList) by demand ;
    demand ← sum of demands in VMList ;
    supply ← pm.getTotalMips();
    while demand >supply do
        violation++;
        demand ← demand -
        vmDemand(VMList[violation]);
    end
    numOfVMsInViolation ←
    numOfVMsInViolation + violation;
end
return(numOfVMsInViolation *
slaViolationCostPerSec * t);
```

Violation cost due to VM migrations

Performance degradation due to migration (PDM) represents the performance degradation due to the overhead resulting from the migration of VMs among PMs. VMs are not available until the end of the migration process, which causes violation to the SLA. Algorithm 4 computes the cost of performance degradation due to VM migration where the *migrationTime* represents the time required until the VM is migrated. The time required to migrate a VM equals the amount of memory used by the VM divided by the available network bandwidth [3].

Algorithm 4: Violation cost due to VM migration

```
float PDMCost(a, t)
Result: Cost of SLA violations (SLAVs) due to
        migration
Initialization:
migrationTime ← 0; pdmViolationCost ← 0;
foreach vm in the Assignment a do
    if the vm is migrated then
        migrationTime ← vm.getAllocatedRam() /
        pm.getBw();
        pdmViolationCost ← pdmViolationCost +
        migrationTime * pdmCostSec;
    end
end
return(pdmViolationCost);
```

Representation design

The solution to the VMs-to-PMs assignment problem is simply represented by a map. This map has *m* elements where each element, key, represents a VM and the value of that element is the PM's ID to which the VM is assigned. Figure 4 displays the representation of the solution.

Optimization algorithm

Finding a good assignment that maximizes the utility, *fitness*, function involves searching candidate assignments. A genetic, evolutionary, algorithm can be used for searching the search space for an appropriate assignment.

Fig. 4 Representation of the solution. Solution representation of the VMs-to-PMs assignment problem

Genetic algorithms are one of the evolutionary computation methods that are considered as powerful stochastic search and optimization techniques based on the Darwinian principle of natural selection [30, 31]. Genetic algorithms search a population of individuals in parallel; therefore they have the ability to avoid locally optimal solutions. Algorithm 5, adopted from [31, 32], shows a high-level pseudo-code of the genetic algorithm used for finding the VM-to-PM assignment that maximizes the utility. The genetic algorithm goes through the following phases: creation of initial population, fitness evaluation, parents selection, crossover, mutation and new population selection.

Algorithm 5: Genetic Algorithm Pseudo-code, modified version of [31, 32]

Result: Best-found VMs-to-PMs assignment
Initialization:
popSize ← population size; genCount ← number of generations;
P ← {}; ▷ *Population: set of VMs-to-PMs assignments*($a_1, a_2, ...a_n$)
P ← initialVmsToPmsAssignments(popSize);
fitnessEvaluation(P); ▷ *based on the utility function defined in "Utility function definition" section*
while *generation number is less than genCount* **do**
  newPop ← {}; ▷ new population for next generation
  for *popSize times* **do**
    parentAssignment a_i, a_j ← parentsSelection(P);
    childrenAssignment c_i, c_j ← crossover(a_i, a_j);
    newPop ← newPop ∪ {mutate(c_i), mutate(c_j)};
  end
  fitnessEvaluation(P, newPop);
  P ← newPopulation(P, newPop);
end
return(best-found VMs-to-PMs assignment)

Initial population

The initial population consists of a number of individuals where each individual is considered a candidate solution to the problem. In the VM placement problem, each individual is described by a map, where a VM represents the *key* and the *value* of that key is the ID of the PM to which the VM is assigned, as shown in "Representation design" section. The *initialVmsToPmsAssignments(popSize)* in Algorithm 5, generates the initial set of solutions, candidate VMs-to-PMs assignments, by mutating the currently working solution. The current VMs-to-PMs assignment is a member of the initial population so that it can be kept if there is nothing better. The number of individuals in the population depends on the *populationSize* parameter.

Fitness/utility function evaluation

Each individual, *candidate solution*, is evaluated against the fitness, *utility*, function introduced in "Utility function definition and Utility model development" sections. A higher fitness means a better result as the goal is to maximize the income due to the VM placement. The fitness function calculates the utility of each individual.

Parents selection

We need to select individuals from the current population to be parents for the crossover operation. There are many selection methods such as *roulette wheel selection*, *random selection*, *tournament selection*, *rank selection*, and *Boltzmann selection* [33]. We have deployed *random selection* by randomly selecting individuals from the current population for crossover.

Crossover

The crossover operator creates new individuals by combining parts of two individuals. Parents need to be selected from the current population to be crossed over. There are various crossover techniques such as *single point crossover*, *multi-point crossover*, *uniform crossover*, *shuffle crossover* and *ordered crossover* [33]. We have deployed a *single point crossover* where swapping between the two individuals is done beyond the crossover point. The crossover operation creates a new population of the crossed over individuals. Table 1, shows how the new offspring is created by mating two parents using single point crossover. The "‖" symbol represents the crossover point.

Mutation

The mutation function creates new individuals by making changes in one or more values in a single individual. These new individuals are similar to current individuals, with

Table 1 Single point crossover

$Parent_1$	$PM_1\ PM_2\ PM_1 \parallel PM_3\ PM_2\ PM_1$
$Parent_2$	$PM_2\ PM_1\ PM_3 \parallel PM_1\ PM_3\ PM_3$
$Offspring_1$	$PM_1\ PM_2\ PM_1 \parallel PM_1\ PM_3\ PM_3$
$Offspring_2$	$PM_2\ PM_1\ PM_3 \parallel PM_3\ PM_2\ PM_1$

changes occurring based on a pre-defined probability. The mutation is used to make changes to the population from one generation to another. A portion of the current population is randomly selected to be mutated. Each individual, VMs-to-PMs assignment, resulting from the selection is mutated using the *mutation probability*. The mutation process creates a new separate population of the mutated individuals. Table 2 shows an example of how the mutation of the PMs works. The PMs in bold in the original offspring are mutated to new ones in the mutated offspring which means that some VMs are assigned to new PMs.

New population selection

Suppose that population size is N VMs-to-PMs assignments. The population of the next generation is created by selecting best K VMs-to-PMs assignment and copying them to the new population where $K < N$. The remaining $N-K$ solutions are randomly copied to complete the population size. This approach ensures that individuals, VMs-to-PMs assignments, with the highest utility are retained, while also maintaining diversity in the population.

The resulting assignment

The genetic algorithm goes into a loop, the number of iterations of this loop is defined by the number of generations. After the end of the specified number of generations, the algorithm selects the individual with the highest fitness from the current population to represent the solution of the problem.

Experimental evaluation

Baseline heuristic method

We compared our utility-based solution against the heuristics-based one proposed by Beloglazov et al. [2, 3]. Beloglazov et al. [2, 3] have carried out a comprehensive investigation of heuristic techniques for energy-aware dynamic VM placement in cloud data centers. They divided the dynamic VM consolidation problem into four sub-problems and proposed different algorithms for solving each of these sub-problems.

Table 2 Mutating the PMs

Original Offspring	$PM_1\ \mathbf{PM_2}\ PM_1 \ldots \mathbf{PM_1}\ \mathbf{PM_3}\ PM_3$
Mutated Offspring	$PM_1\ PM_4\ PM_1 \ldots PM_2\ PM_2\ PM_3$

The first sub-problem is *host overload detection* and involves deciding when a specific host PM is considered to be overloaded. Host overload detection requires migrating one or more VMs from the overloaded host to a non-overloaded one. They developed three main solutions for solving the host overload detection problem. These solutions are using a *static CPU utilization threshold*, an *adaptive threshold* using median absolute deviation (MAD) and interquartile range (IQR), and *local regression-based* using local regression (LR) and robust local regression (LRR). They found that the LRR algorithm was the best for solving the host overload detection problem. The second sub-problem was host under-load detection, which involves identifying when to decide that a host PM is under-loaded. They migrate all VMs from an under-loaded host to other hosts, if possible, and then switch that machine into power saving mode. The third sub-problem was VM selection and it involves identifying which VMs should be selected from an overloaded host for migration. They proposed 3 VM selection policies, namely minimum migration time (MMT), random selection (RS) and maximum correlation (MC). They found that *MMT* was the best for VM selection. The final sub-problem was the VM placement problem, and it involves identifying which hosts should be used for placing the migrated VMs. They deployed a power aware best fit decreasing (PABFD) algorithm, which is a modified version of the best fit decreasing (BFD) that considers CPU utilization during the allocation.

According to their evaluations, *LRR* and *MMT* were the best solutions for host overload detection and VM selection problems respectively. Therefore, we compare our proposed solution against the heuristics-based one that uses LRR and MMT.

Performance metrics

Four performance metrics are used for evaluating the effectiveness of the proposed VM placement approach and comparing it with the baseline heuristic method. These performance metrics are energy consumption, overall SLA violations, the number of migrations and the number of PM shutdowns.

- Energy Consumption: represents the total amount of energy consumed by all running PMs in the cloud data center. Lower values of energy consumption help reduce expenditure. This means that the lower the amount of energy consumed, the better the assignment.
- Overall SLA Violations: The SLA violations occur either due to the over-utilization of the PM or due to the migration of VMs among PMs. The lower the overall SLA violations, the better the assignment. The overall SLA violations are represented as a

percentage and calculated as follows:

$$overallSlaViolations = \frac{totalRequestedMips - totalAllocatedMips}{totalRequestedMips}$$

where *totalRquestedMips* is the totals MIPS requested by all VMs and *totalAllocatedMips* is the actual total MIPS allocated to the VMs based on the resource demand.

- Number of Migrations: This metric shows how often the VMs are migrated among PMs. Too many migrations degrade the performance and increase the SLA violations, and too few migrations might lead to an inappropriate assignment, and hence a balance is required to consider this trade-off.
- Number of PM Shutdowns: This metric indicates how many times the PMs are shut down. The frequent switching of a PM on and off might lead to PM failure on the long run [34].
- Profit: The profit metric calculates the average profit per day. The profit is calculated using: $profit = incomeFromVMs - (eneryCost + overallViolationCost)$. It ignores the PMs' cost, and the profit is highly dependent on factors such as the specifics of SLA violations and energy costs.

Simulation environment

The CloudSim simulation toolkit is used for simulating and evaluating the proposed utility-based approach [35, 36]. One cloud data center is simulated using two different types of PM, namely, *HP ProLiant ML110 G4* (Intel Xeon 3040, 2 cores x 1860 MHz, 4096 MB RAM and 1 Gbps BW) and *HP ProLiant ML110 G5* (Intel Xeon 3075, 2 cores x 2660 MHz, 4096 MB RAM and 1 Gbps BW). The cloud data center hosts four different VM types namely large, medium, small and extra small. The four types of the VMs are *large instance* (1 core x 2500 MHz, 870 MB RAM and 100 Mbps BW), *medium instance* (1 core x 2000 MHz, 1740 MB RAM and 100 Mbps BW), *small instance* (1 core x 1000 MHz, 1740 MB RAM and 100 Mbps BW) and *extra-small instance* (1 core x 500 MHz, 613 MB RAM and 100 Mbps BW).

Every VM runs an application with a different workload. The Applications' workloads are represented by CloudSim Cloudlets [36]. The Cloudlets randomly generate utilization data every 5 minutes based on a stochastic model [3]. The average percentage of utilization is approximately 50 %. All the following experiments have been run on an HP Pavilion g6 laptop(Core i7, 6 GB RAM).

Experiment setup

The simulation lasts for one day of simulation time which is the same time used in the heuristics-based solution. The algorithm runs every 5 minutes, 300 seconds, which is the same interval used in VMware's distributed resource scheduler (DRS) [37]. We have used a *population* of 20 individuals, the *number of generations* was 40, the *crossover ratio* was 0.8 and the *mutation probability* was 0.7. The total number of physical machines used in the conducted experiments is 50 and 100 PMs. Moreover, the total number of VMs used range from 50 to 200 VMs.

Two experiments with three different configurations for each were conducted to evaluate the proposed solution. The goal of the first experiment is to test the effectiveness of the solution on lightly loaded data centers while the second experiment tests the solution in more loaded cloud data centers. These scenarios are considered relevant because we need to be confident that: (i) the opportunities that exist to save energy are exploited in lightly loaded settings, and (ii) adaptation costs and attempts at energy saving do not lead to more SLAs being missed in a heavily loaded setting. Eventually, an experiment has been conducted to measure the execution time and compare the profit gained from applying both the utility-based and the heuristics-based solutions.

Experiments descriptions and results

Experiment 1: lightly loaded cloud data centres

Experiment 1 aims to appraise the effectiveness of the proposed utility based solution in a lightly loaded data center. In this experiment, the number of VMs is the same as the number of PMs that they will be allocated to. Three different configurations have been chosen for testing lightly loaded data centers:

1. Configuration 1.1: 50 VMs allocated to 50 PMs.
2. Configuration 1.2: 100 VMs allocated to 100 PMs.
3. Configuration 1.3: 150 VMs allocated to 150 PMs.

Figure 5 shows a scatter plot that presents the results of running *Configuration 1.1*. Each point in the scatter plot shows the results for a run of a workload; the vertical axis reports the energy consumed by the run and the horizontal axis shows the percentage of time when SLAs are not met. Thus a position in the bottom left of the chart is best. The blue triangles are results for the utility-based approach and the orange dots for the heuristic approach. The utility-based solution reduced energy consumption on average by about 10 % and reduced the time in which SLAs were being violated by around 29 %.

Figure 6 demonstrates the results of running *Configuration 1.2*. The utility based solution reduced average energy consumption by about 5 % and the time in which SLAs were being violated on average by around 38 %.

Figure 7 exhibits the results of running *Configuration 1.3*. The utility based solution improved energy consumption by about 5 % and reduced the time in which SLAs were being violated on average by about 40 % compared to the results of the heuristics based solution.

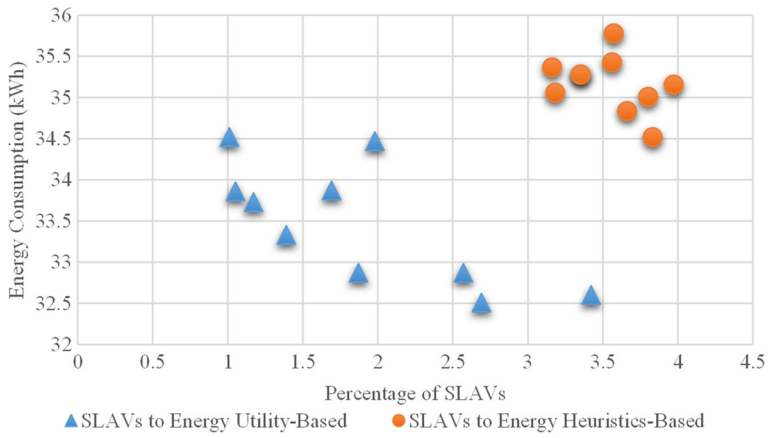

Fig. 5 Overall SLA violations to energy consumption from running *Configuration 1.1*, 10 times. The *blue triangles* represent results from the utility-based solution while *orange circles* represent the heuristics-based one

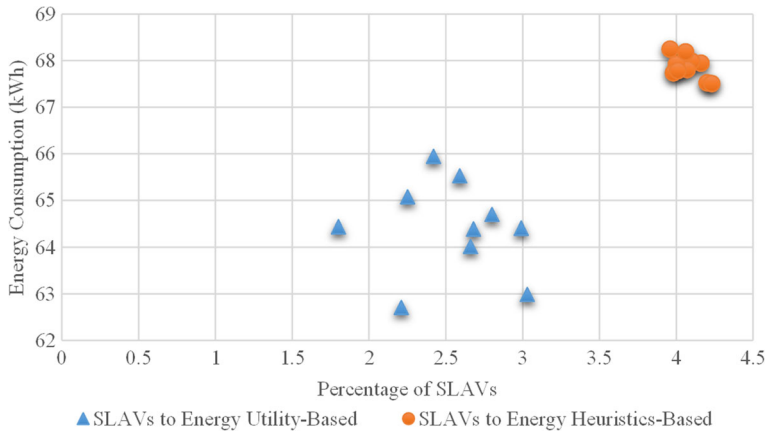

Fig. 6 Overall SLA violations to energy consumption after running *Configuration 1.2*, 10 times. The *blue triangles* depict results from the utility-based approach while *orange circles* represent the heuristics-based one

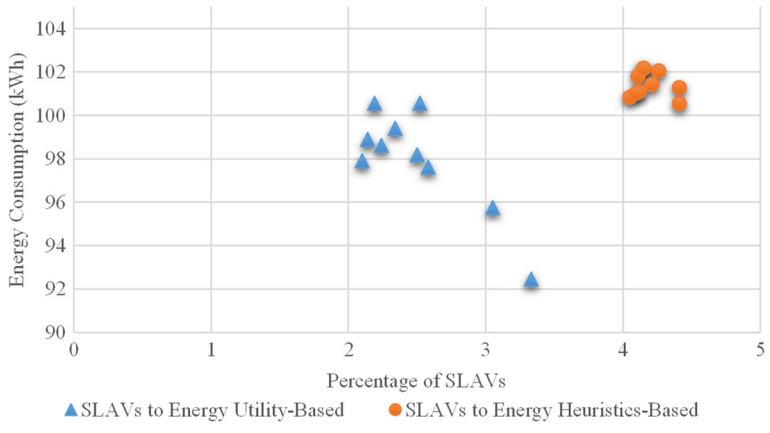

Fig. 7 Overall SLA violations to energy consumption after running *Configuration 1.3*, 10 times. The *blue triangles* represent results from the utility-based approach while *orange circles* represent the heuristics-based one

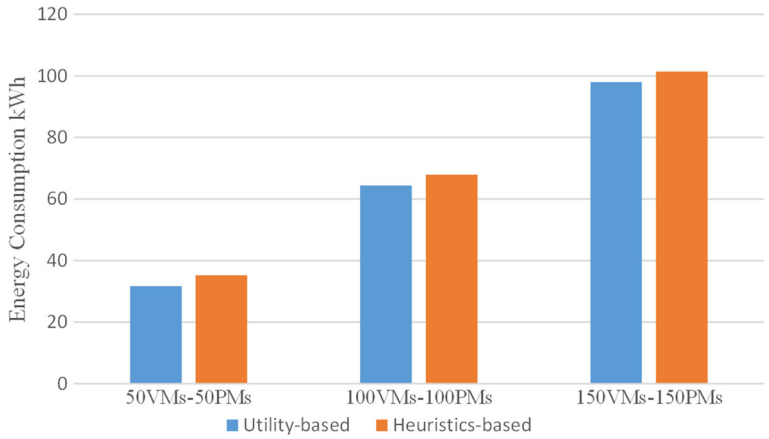

Fig. 8 Average energy consumption, from *Experiment 1*. The *blue* columns represent average energy consumption from the utility-based solution while the *orange* ones from the heuristics-based

For lightly-loaded data centers, the proposed utility-based solution outperforms the heuristics-based one in terms of energy savings and reduction of SLA violations, as summarized in the average values of *Experiment 1* in Figs. 8 and 9. Furthermore, the average values of *Experiment 1*, shown in Figs. 10 and 11, demonstrate that the utility-based solution saves energy consumption and reduces the overall SLAVs using a smaller number of VMs migrations and PMs shutdowns. Experiment 1 shows an average saving in energy consumption in lightly loaded data centers of approximately 5 % and an average reduction in SLA violations of around 36 %. Finally, the average number of VMs migrations and PMs shutdowns in the utility-based solution represent only around 16 and 5 % respectively, of the migrations and PMs shutdowns in the heuristics-based one.

These reductions in the number of VM migrations and PM shutdowns result from the fact that the utility-based approach only migrates when it finds a VM-to-PM assignment for which an adaptation is expected to be beneficial. In contrast, the heuristics based approach migrates whenever there is a problem (i.e. host/PM overload or host under-load); however, a problem may be detected without there being a solution to which it is worthwhile to adapt, and hence the heuristic approach tends to over-adapt.

Experiment 2: more loaded cloud data centres

Experiment 2 aims to assess the impact of increasing the number of VMs per PM on the specified performance metrics. The three configurations used for testing the more loaded data centers are as follows:

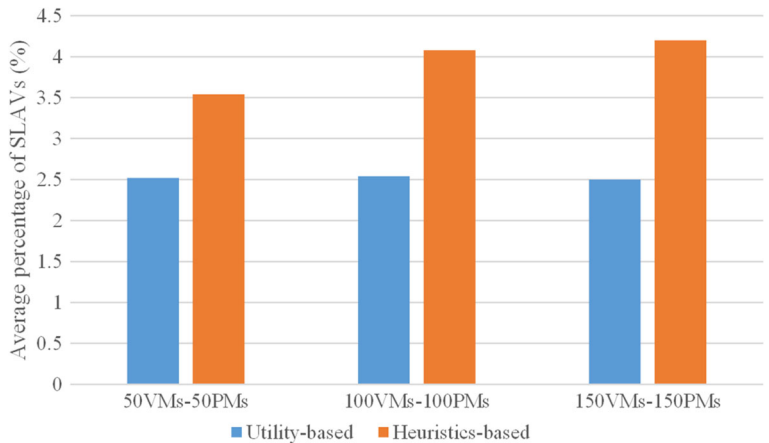

Fig. 9 Average SLA violations from *Experiment 1*. The *blue* columns represent average number of the overall SLAVs from the utility-based approach while the *orange* ones from the heuristics-based

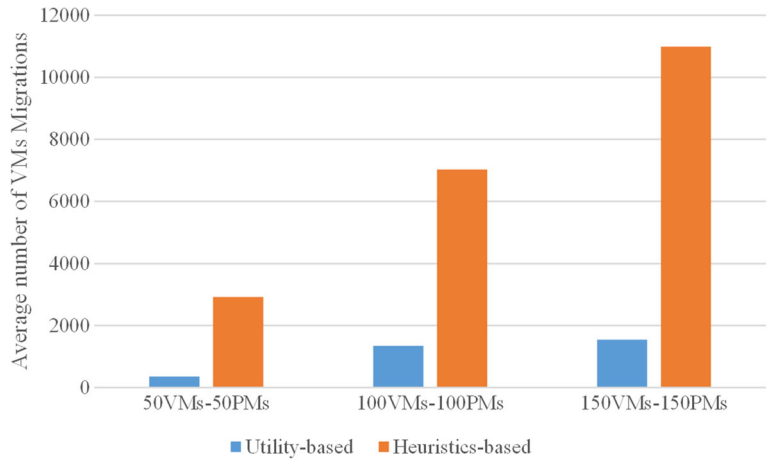

Fig. 10 Average number of VM migrations from *Experiment 1*. The *blue* columns represent the average number of VM migrations from the utility-based approach while the *orange* ones from the heuristics-based

1. Configuration 2.1: 100 VMs allocated to 100 PMs.
2. Configuration 2.2: 150 VMs allocated to 100 PMs.
3. Configuration 2.3: 200 VMs allocated to 100 PMs.

The results of running *Configuration 2.1* are the same as those for *Configuration 1.2* in "Experiment 1: lightly loaded cloud data centres" section, as they have the same configurations. Figure 12 shows a comparison between energy consumption and overall SLAVs after running *Configuration 2.2*. The utility-based solution reduced the average energy consumption and SLA violations by around 6 and 42 %, respectively compared to the heuristics based one. Comparing the results from *Configuration 2.1* and *Configuration 2.2*, we can conclude that increasing the number of VMs by 50 % using the utility based approach resulted in nearly the same percentage of overall SLA violations, while the energy consumption is increased by about 50 %. However, the same increase in the number of VMs using the heuristics based approach also results in an increase in the percentage of the time spent in violations by about 6 % while the energy consumption is also increased by about 50 %. This means that the utility-based approach is more consistent in terms of SLAVs in more loaded data centers. However, there is slightly more performance degradation in the case of the heuristics based approach.

The scatter plot in Fig. 13 shows the values of energy consumption and overall SLAVs using both solutions after running *Configuration 2.3*. The utility based solution reduced energy consumption by about 6 %. Moreover, the

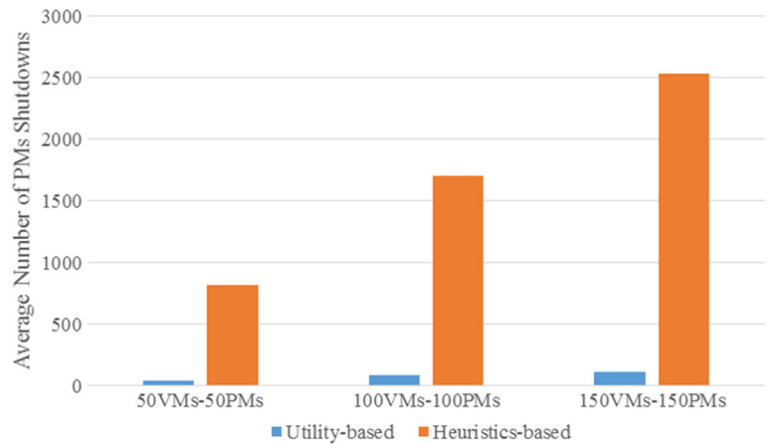

Fig. 11 Average number of PM shutdowns from *Experiment 1*. The *blue* columns represent the average number of PM shutdowns from the utility-based approach while the *orange* ones from the heuristics-based

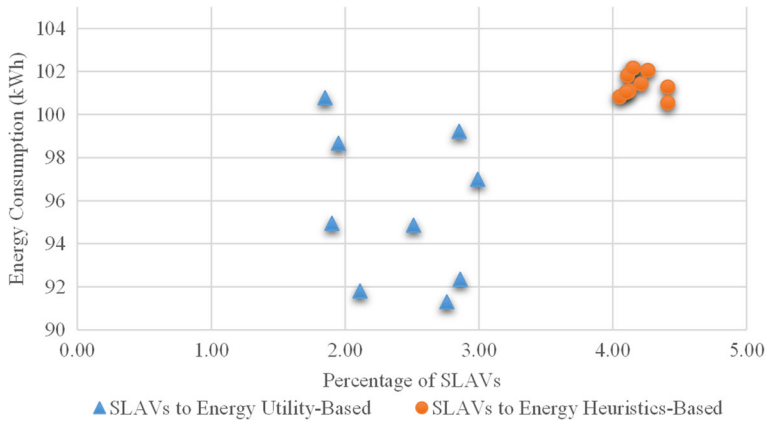

Fig. 12 Overall SLA violations to energy consumption from running *Configuration 2.2*, 10 times. The *blue triangles* represent the utility-based approach while the *orange circles* represent the heuristics-based one

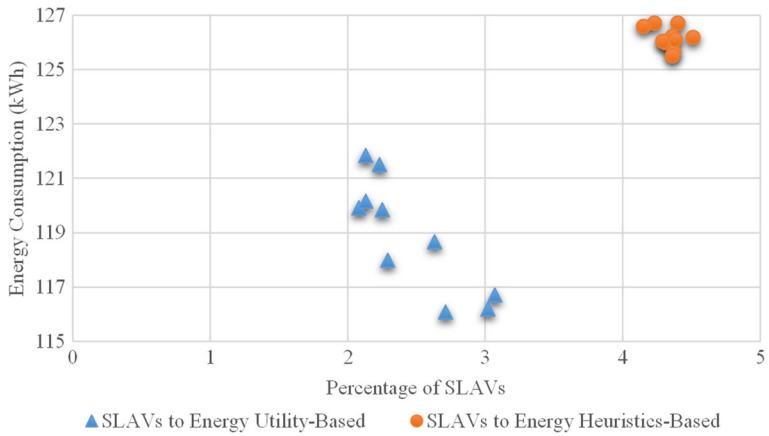

Fig. 13 Overall SLA violations to energy consumption from running *Configuration 2.3*, 10 times. The *blue triangles* represent the proposed utility-based solution while the *orange circles* represent the heuristics-based one

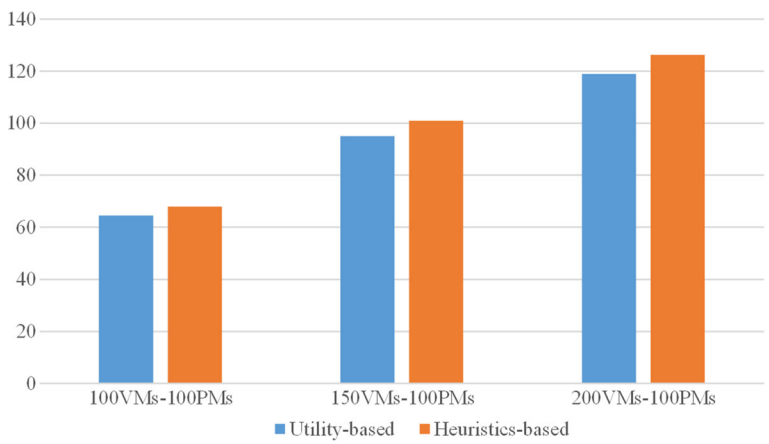

Fig. 14 Average energy consumption from *Experiment 2*. The *blue* columns represent average energy consumption from the utility-based approach while the *orange* ones from the heuristics-based

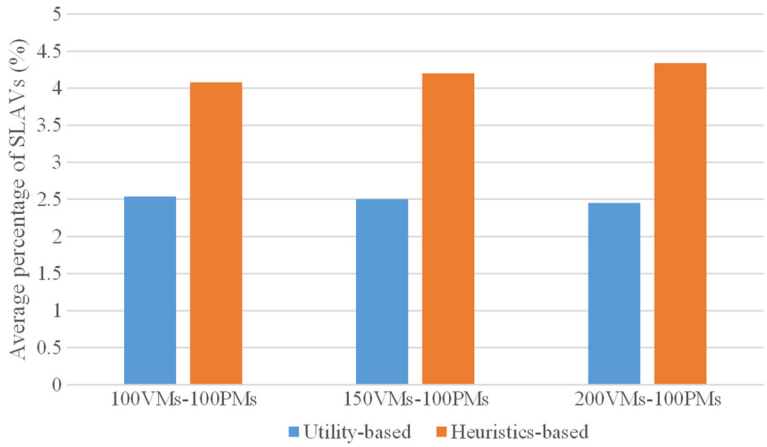

Fig. 15 Average SLA violations from *Experiment 2*. The *blue* columns represent average number of the overall SLAVs from the utility-based approach while the *orange* ones from the heuristics-based

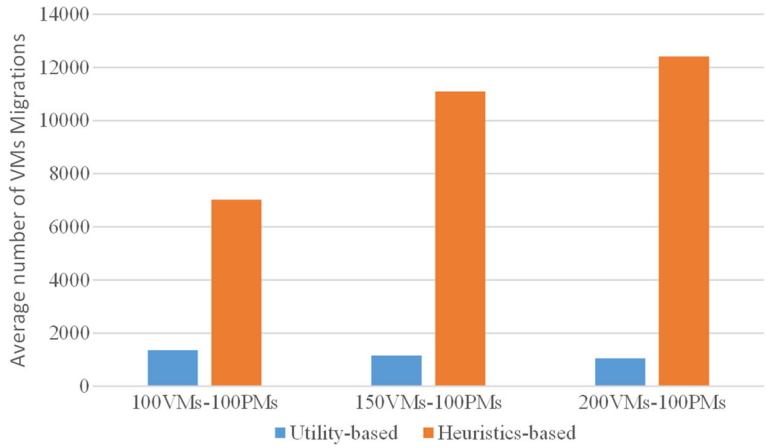

Fig. 16 Average number of VM migrations from *Experiment 2*. The *blue* columns represent the average number of VM migrations from the utility-based approach while the *orange* ones from the heuristics-based

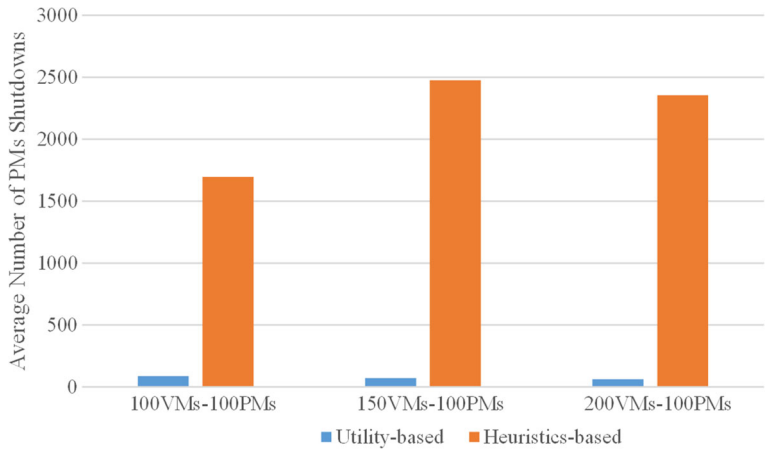

Fig. 17 Average number of PM shutdowns from *Experiment 2*. The *blue* columns represent the average number of PM shutdowns from the utility-based approach while the *orange* ones from the heuristics-based

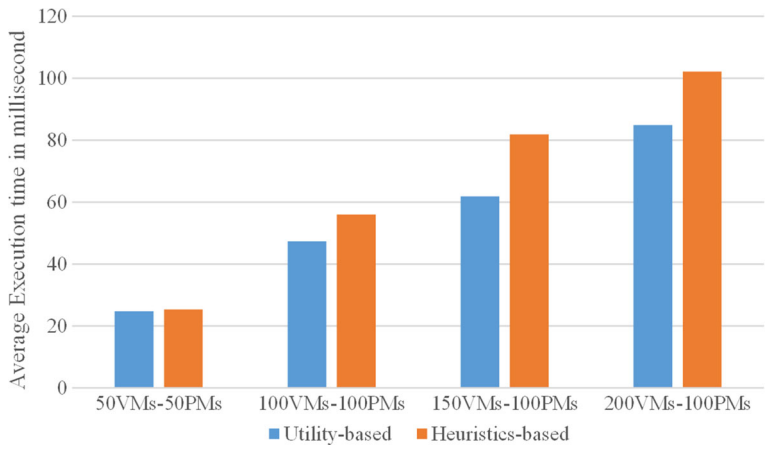

Fig. 18 Average execution time. Each column represents the average execution time; *blue* columns represent the utility-based approach, while the *orange* ones from the heuristics-based

utility based solution reduced time spent in SLA violations by about 43 %.

Figures 14, 15, 16 and 17 confirm that the proposed utility-based solution is also better than the heuristics-based one as it can save more energy and reduces the overall SLAVs using a smaller number of migrations and PM shutdowns. Experiment 2 shows that the utility-based solution reduces the amount of energy consumed in more-loaded data centers by around 6 % and also reduces the average SLA violations by nearly 41 %. Moreover, the average number of VMs migrations and PMs shutdowns in the utility-based solution represent approximately 12 and 3 % respectively, of the migrations and PMs shutdowns in the heuristics-based one.

Results conclude that the proposed utility-based solution saves more energy and reduces SLA violations more effectively than the heuristics-based one both in heavily and lightly loaded data center settings.

Experiment 3: average execution time and profit

Experiment 3 measures the average execution time and the average profit per day for both solutions. The time complexity of the genetic algorithm is O(gnm) + the complexity of the fitness (utility) function; where g is the number of generations, n is the population size and m is the solution size (number of VMs). Figure 18 exhibits the average execution time of both approaches. It demonstrates that the proposed utility-based approach can make a better VMs-to-PMs assignment in less execution time.

The profit is calculated based on the profit metric defined in "Performance metrics" section and the parameters used as shown in Table 3; based on the VM pricing offered by Amazon EC2 pricing[1]. Figure 19 confirms that the average profit per day from the utility-based approach outperforms the average profit from the heuristics-based approach.

Conclusions and further work

Conclusion

This paper presented an approach based on utility functions for creating a self-managing VM placement solution in cloud data centers that dynamically assigns VMs-to-hosts according to resource utilization. The main goal of the approach is to increase the profit of an IaaS provider by minimizing the cost of energy consumption and the cost of different sources of SLAVs. Experiments have been conducted for comparing the effectiveness of the proposed utility based solution with an existing heuristic-based solution. The heuristic method against which the comparison took place was subjected by its proposers to a systematic evaluation in comparison with alternative heuristics, and shown to perform well [2, 3]. The empirical evaluation uses the original authors' implementation of the heuristic approach.

Evaluation showed that the proposed utility based solution outperformed the existing heuristic based approach in terms of energy savings and minimizing SLAVs in both lightly loaded and more heavily loaded cloud data centers. Perhaps the key factor that differentiates the approaches is that the heuristics based approach adapts whenever there is a problem (*PM overload, or PM under-load*). On the contrary, the utility based approach adapts only if it can identify an adaptation that is expected to improve on the current allocation.

Table 3 Profit parameters

VM_{large}	VM_{medium}	VM_{small}	VM_{XSmall}	Energy	Overall SLAVs
0.293 $/h	0.146 $/h	0.073 $/h	0.028 $/h	0.11 $/h	0.4 $/h

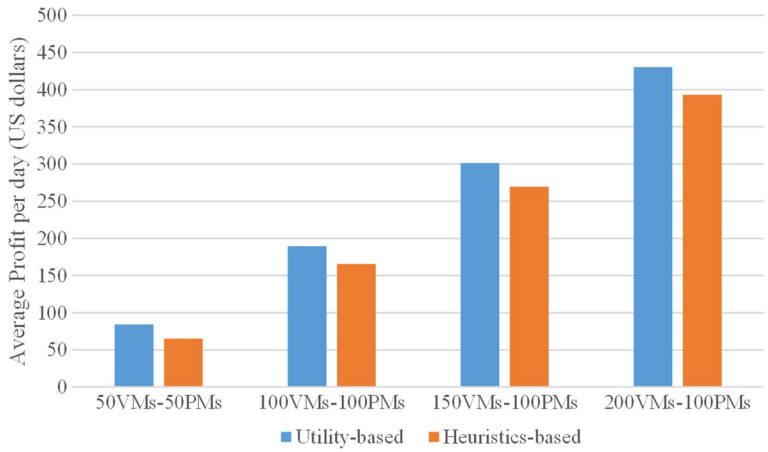

Fig. 19 Average profit per day. The *blue* columns depict the average profit/day from the utility-based approach, while the *orange* ones from the heuristics-based

Future directions

Although the proposal as described has shown considerable promise in empirical evaluations, the following points represent some areas that may benefit from further research:

1. Improving the utility model:
   More work is required to refine the utility model. For example, the calculations of the SLA violations could be improved by implementing a proactive calculation of the expected CPU utilization instead of the reactive one deployed in the paper.
2. Considering memory and network I/O during VM placement:
   In this paper, we only considered CPU during VM placement. Although the CPU consumes most of the server's power, all other host and network resources should be involved in the adaptive VM-to-PM assignment. For example, memory and network I/O should be considered during VM placement as they have a significant effect in memory and network intensive applications.
3. Revisiting the search algorithm:
   We have used a genetic algorithm for exploring the search space to find an efficient assignment that maximizes the utility. However, alternative approaches could be explored, such as the use of multi-dimensional optimization techniques that optimize for SLA violations and energy as distinct dimensions.
4. Scalability with real workload traces:
   Building a scalable VM placement strategy still requires further research for finding the most suitable scaling technique. Moreover, we need to conduct experiments with different workloads and different sizes of Cloud data centres.
5. Considering multi-tenancy constraints:
   The VM placement solution should consider multi-tenancy constraints such as security, anti-colocation and reducing network latency between VMs belonging to the same user.
6. Considering other sources of violations in virtualized environments:
   In our proposed solution, we only considered two sources of violations, namely performance degradation due to migration and the over-utilization of PMs. However, there are many other sources of violations such as violations resulting from the interference among collocated workloads, system outage, and late-time failure.

Endnote

[1] https://aws.amazon.com/ec2/pricing.

Authors' contributions

AM carried out the detailed design, implementation and evaluation of the work. NP conceived the study, and participated in its design and coordination. Both authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Received: 11 December 2015 Accepted: 15 October 2016
Published online: 24 October 2016

References

1. Jennings B, Stadler R (2015) Resource management in clouds: Survey and research challenges. J Network Syst Manage 23(3):567–619. doi:10.1007/s10922-014-9307-7
2. Beloglazov A, Abawajy JH, Buyya R (2012) Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Futur Gener Comput Syst 28(5):755–768. doi:10.1016/j.future.2011.04.017
3. Beloglazov A, Buyya R (2012) Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurr Comput Pract Experience 24(13):1397–1420. doi:10.1002/cpe.1867

4. Beloglazov A, Buyya R (2013) Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints. IEEE Trans Parallel Distrib Syst 24(7):1366–1379. doi:10.1109/TPDS.2012.240
5. Walsh WE, Tesauro G, Kephart JO, Das R (2004) Utility functions in autonomic systems. In: Autonomic Computing, 2004. Proceedings. International Conference On. IEEE. pp 70–77
6. Kephart JO, Das R (2007) Achieving self-management via utility functions. IEEE Internet Comput 11(1):40–48
7. Paton NW, Buenabad-Chavez J, Chen M, Raman V, Swart G, Narang I, Yellin DM, Fernandes AAA (2009) Autonomic query parallelization using non-dedicated computers: an evaluation of adaptivity options. VLDB J 18:119–140
8. Lee K, Paton NW, Sakellariou R, Fernandes AAA (2011) Utility functions for adaptively executing concurrent workflows. Concurr Comput Pract Experience 23(6):646–666. doi:10.1002/cpe.1673
9. Das R, Kephart JO, Lenchner J, Hamann HF (2010) Utility-function-driven energy-efficient cooling in data centers. In: Proceedings of the 7th International Conference on Autonomic Computing, ICAC 2010, Washington, DC, USA, June 7-11, 2010. pp 61–70. doi:10.1145/1809049.1809058. http://doi.acm.org/10.1145/1809049.1809058
10. Beloglazov A, Buyya R, Lee YC, Zomaya AY (2011) A taxonomy and survey of energy-efficient data centers and cloud computing systems. Adv Comput 82:47–111. doi:10.1016/B978-0-12-385512-1.00003-7
11. Lorido-Botran T, Miguel-Alonso J, Lozano JA (2014) A review of auto-scaling techniques for elastic applications in cloud environments. J Grid Comput 12(4):559–592. doi:10.1007/s10723-014-9314-7
12. Shi L, Butler B, Botvich D, Jennings B (2013) Provisioning of requests for virtual machine sets with placement constraints in iaas clouds. In: 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), Ghent, Belgium, May 27-31, 2013. pp 499–505
13. Rabbani MG, Pereira Esteves R, Podlesny M, Simon G, Zambenedetti Granville L, Boutaba R (2013) On tackling virtual data center embedding problem. In: Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium On. IEEE. pp 177–184
14. Wolke A, Tsend-Ayush B, Pfeiffer C, Bichler M (2015) More than bin packing: Dynamic resource allocation strategies in cloud data centers. Inf Syst 52:83–95
15. Borgetto D, Stolf P (2014) An energy efficient approach to virtual machines management in cloud computing. In: Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference On. IEEE. pp 229–235
16. Monil MAH, Qasim R, Rahman RM (2014) Energy-aware VM consolidation approach using combination of heuristics and migration control. In: Ninth International Conference on Digital Information Management, ICDIM. pp 74–79. doi:10.1109/ICDIM.2014.6991413. http://dx.doi.org/10.1109/ICDIM.2014.6991413
17. More D, Metha S, Walase L, Abraham J (2014) Achieving energy efficiency by optimal resource utilisation in cloud environment. In: IEEE Intl. Conf. on Cloud Computing in Emerging Markets (CCEM). pp 1–8
18. Xiao Z, Song W, Chen Q (2013) Dynamic resource allocation using virtual machines for cloud computing environment. IEEE Trans Parallel Distrib Syst 24(6):1107–1117
19. Xu J, Fortes J (2011) A multi-objective approach to virtual machine management in datacenters. In: Proceedings of the 8th ACM International Conference on Autonomic Computing. ACM. pp 225–234
20. Setzer T, Wolke A (2012) Virtual machine re-assignment considering migration overhead. In: Network Operations and Management Symposium (NOMS), 2012 IEEE. IEEE. pp 631–634
21. Monil MAH, Rahman RM (2016) Vm consolidation approach based on heuristics fuzzy logic, and migration control. J Cloud Comput 5(1):1–18
22. Chowdhury MR, Mahmud MR, Rahman RM (2015) Implementation and performance analysis of various vm placement strategies in cloudsim. J Cloud Comput 4(1):1
23. Beloglazov A, Buyya R (2010) Energy efficient resource management in virtualized cloud data centers. In: Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing. IEEE Computer Society. pp 826–831
24. Bennani MN, Menasce D, et al (2005) Resource allocation for autonomic data centers using analytic performance models. In: Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference On. IEEE. pp 229–240
25. Koehler M (2014) An adaptive framework for utility-based optimization of scientific applications in the cloud. J Cloud Comput 3(1):1–12
26. Paton NW, Aragão MAT, Fernandes AAA (2012) Utility-driven adaptive query workload execution. Futur Gener Comp Syst 28(7):1070–1079. doi:10.1016/j.future.2011.08.014
27. Paton N, De Aragão MA, Lee K, Fernandes AA, Sakellariou R (2009) Optimizing utility in cloud computing through autonomic workload execution. Bull Tech Comm Data Eng 32(1):51–58
28. Ferdaus MH, Murshed M, Calheiros RN, Buyya R (2014) Virtual machine consolidation in cloud data centers using aco metaheuristic. In: European Conference on Parallel Processing. Springer. pp 306–317
29. Moreno IS, Yang R, Xu J, Wo T (2013) Improved energy-efficiency in cloud datacenters with interference-aware virtual machine placement. In: Autonomous Decentralized Systems (ISADS), 2013 IEEE Eleventh International Symposium On. IEEE. pp 1–8
30. Whitley D (1994) A genetic algorithm tutorial. Stat Comput 4(2):65–85
31. Luke S (2013) Essentials of metaheuristics. http://cs.gmu.edu/~sean/book/metaheuristics/Essentials.pdf. Accessed 1 Aug 2016
32. Alcaraz J, Maroto C (2001) A robust genetic algorithm for resource allocation in project scheduling. Ann Oper Res 102(1-4):83–109
33. Sivanandam S, Deepa S (2007) Introduction to Genetic Algorithms. Springer Science & Business Media
34. Monil MAH, Qasim R, Rahman RM (2014) Energy-aware vm consolidation approach using combination of heuristics and migration control. In: Digital Information Management (ICDIM), 2014 Ninth International Conference On. IEEE. pp 74–79
35. Calheiros RN, Ranjan R, De Rose CA, Buyya R (2009) Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services. arXiv preprint arXiv:0903.2525. cloudbus.cis.unimelb.edu.au/reports/CloudSim-ICPP2009.pdf. Accessed 21 Oct 2016
36. Calheiros RN, Ranjan R, Beloglazov A, De Rose CA, Buyya R (2011) Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw Pract Experience 41(1):23–50
37. Gulati A, Shanmuganathan G, Holler A, Ahmad I (2011) Cloud-scale resource management: challenges and techniques. In: Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing. USENIX Association. pp 3–3