Received: 13 June 2019 | Revised: 12 May 2020 | Accepted: 23 November 2020

DOI: 10.1002/cpe.6138

RESEARCH ARTICLE

WILEY

Comparison of workload consolidation algorithms for cloud data centers

René Ponto[1] | Gábor Kecskeméti[2] | Zoltán Á. Mann[1]

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

[2]Department of Computer Science, Liverpool John Moores University, Liverpool, UK

Correspondence
Zoltán Á. Mann, paluno – The Ruhr Institute for Software Technology, University of Duisburg-Essen, Essen, Germany.

Funding information
Horizon 2020 Framework Programme, Grant/Award Number: 731678

Abstract

Workload consolidation is an important method for the efficient operation of cloud data centers, impacting important quality attributes such as resource utilization and power consumption. Many different approaches have been proposed for workload consolidation, but few comparative studies were executed to date. Therefore, it is unclear which of the proposed approaches work best in which situation. In this article, we present a comprehensive simulation-based comparison of five workload consolidation techniques. We introduce a general framework for workload consolidation techniques to the DISSECT-CF simulator to foster the development and comparison of efficient data center consolidation algorithms. We use this framework to evaluate the effectiveness of a first fit best fit decreasing heuristic, a custom heuristic, and three population-based metaheuristics (genetic algorithm, artificial bee colony, and particle swarm optimization). The evaluation is based on a wide variety of real-world workload traces. The five algorithms are compared in terms of total energy consumption, the duration of the simulation, and the number of migrations. Based on the results, there is no generally best consolidation technique. The results deliver insight into the pros and cons of the algorithms as well as the impact of different parameters. In particular, the results show that population-based metaheuristics do not offer a significant gain in terms of solution quality to compensate for the increased simulation time.

KEYWORDS

cloud computing, data center consolidation, IaaS, simulation, VM consolidation

1 | INTRODUCTION

Cloud computing enables the virtualized management and provisioning of software and hardware solutions, including computing and storage resources and application runtimes.[1] The elasticity of Infrastructure as a Service (IaaS) clouds is a particularly appealing feature of cloud computing. To provide this elasticity, the data centers underlying an IaaS cloud must effectively cope with a continuously varying workload. By consolidating the workload to as few physical servers as possible, an IaaS provider can achieve good utilization of the available hardware resources and minimize energy consumption by switching off unused servers.[2] In virtualized data centers, workload consolidation is possible as virtual machines (VMs) can be easily migrated between physical machines (PMs), incurring only an acceptable overhead. On the other hand, it is also important to avoid too aggressive consolidation that would lead to overloaded PMs, which in turn may result in violations of service level objectives and performance degradation for the applications hosted in the VMs.[3]

Therefore, the consolidation of VMs is an important optimization problem. It is an intrinsically multicriteria optimization problem, in which a good trade-off between multiple, often conflicting goals has to be found.[4] Relevant optimization objectives include the number of used PMs, total energy consumption, the number of overloaded PMs, and the number of migrations. Most practical formulations of the problem are NP-hard.[5] On the other hand, VM consolidation must be quick, so that a timely reaction to changes in the workload is possible.

Many algorithms have been proposed to solve the VM consolidation problem (VMCP).[6] Since the problem is NP-hard but solutions are needed quickly, most of the proposed algorithms are heuristics, that is, there is no formal guarantee that the results would be optimal. In such cases, the empirical evaluation of the algorithms is especially important to assess their practical applicability. Early solutions for VM consolidation usually relied on simple and fast heuristics such as greedy algorithms, first fit, or best fit (BF).[7,8] In recent years, several researchers suggested to adopt population-based metaheuristics instead, which work with multiple possible solutions and deliver the best one found. Examples of such algorithms include genetic algorithms (GAs),[9] artificial bee colony (ABC),[10] and particle swarm optimization (PSO).[11] Compared with the simpler, single-solution heuristics, population-based metaheuristics are typically characterized by higher execution time, but may be able to deliver better solutions, for example, by being better able to escape from local optima.

The plethora of available algorithms is a challenge for practitioners, because it is not clear which algorithm to choose, for two reasons. On one hand, different articles address slightly different versions of the VMCP (e.g., they consider different objectives), which makes their comparison difficult. On the other hand, the evaluation of the proposed algorithms in the respective articles is often unsatisfactory, for example, using only synthetic data or a single workload. In addition, authors tend to compare their algorithms to trivial competitors (e.g., algorithms that do not take into account some important aspect of the problem).[6]

For these reasons we believe there is a need for a comprehensive evaluation and unbiased comparison of VM consolidation algorithms, using various realistic workloads. This is the main objective of this article. Particular emphasis is placed on population-based metaheuristics, because previous work on comparing VM consolidation algorithms focused on simple single-solution algorithms.[6,12] We are interested in finding out if (i) the execution time penalty of population-based metaheuristics really pays off in the form of significantly improved results and (ii) how different population-based metaheuristics perform compared with each other.

There are two principal ways to empirically compare VM consolidation algorithms: either in a real data center or using simulation. Although using a real data center would offer the most realistic results, there are several obstacles to the practical realization of such experiments, including the access to a real data center and the endangering of the safe operation of the data center with VM consolidation algorithms that are still at a test stage. Simulation has several advantages: the results can be safely reproduced by any researcher with access to only limited hardware resources, and it allows us to easily experiment with different hardware configurations, without the limitations of a specific infrastructure. Moreover, simulation can be much faster than reality, so that, within a given time frame, simulation can lead to more insights.

We chose the simulator DISSECT-CF as the vehicle for our simulations. DISSECT-CF is a mature cloud simulator; in particular, it has been shown to provide faster simulation and more accurate results than the popular CloudSim simulator.[13] At the beginning of this work, DISSECT-CF did not provide specific support for VM consolidation algorithms, so we extended DISSECT-CF accordingly as part of this work. We did this in a generic way and with clear interfaces so that further VM consolidation algorithms can be easily integrated into DISSECT-CF (without the issues that were previously detected in the way VM consolidation is integrated into CloudSim[14]) and will benefit from easy experimentation.

The article makes the following contributions:

- Implementation of a generic framework for integrating VM consolidation algorithms into the simulator DISSECT-CF. In particular, this framework provides appropriate abstractions so that VM consolidation algorithms can carry out tentative migrations of VMs and assess the effect of these migrations without actually performing migrations in the simulator. Moreover, when the VM consolidation algorithm finishes, the best solution it found is automatically implemented by the framework, executing the necessary actions (switching PMs on or off, migrating VMs) in the correct order.
- Implementation of five different VM consolidation algorithms: the first fit heuristic, a custom heuristic, and three population-based metaheuristics (GA, ABC, and PSO). To foster comparability, all algorithms solve the same version of the VMCP, and all population-based metaheuristics use the same solution encoding, the same kinds of data structures, and the same search improvement techniques (as much as this is possible), so that only the actual algorithmic differences remain.
- Comprehensive empirical evaluation of the five algorithms using a variety of real-world workload traces. The evaluation focuses on selecting the best parameter configuration for each VM consolidation algorithm and then on comparing these best configurations with each other. The comparison (both between parameter configurations and between different algorithms) is challenging as several important metrics (energy consumption, simulation time, number of migrations) must be considered. Moreover, the results depend heavily on the used workload trace.

The structure of the article is the following. First, in Section 2, we discuss the state-of-the-art in the field. In Section 3, we describe the generic consolidation framework implemented in the DISSECT-CF simulator. The used algorithms are discussed in detail in Section 4. Section 5 presents the experiments with the extended simulator and the achieved measurement results. Finally, Section 6 concludes the article.

2 | RELATED WORK

VM migration and dynamic voltage and frequency scaling methods are generally used to achieve server consolidation, which help to achieve resource management goals such as load balancing and power management, though they also affect application performance.[15] The unpredictable nature of workloads and the inability to accurately predict application demands call for dynamic, lightweight and adaptive VM migration designs to improve application performance.[15] VM placement and migration are the major challenging issues in management of virtualized data-centers, and many proposals apply different approaches ranging from linear programming, to GAs.[16] Multiobjective proposals can reduce performance and increase the problem complexity, therefore innovative solutions are needed to deal with multiple and complex aims. Our proposed simulation environment aims at providing a way for investigating different algorithms to achieve these goals.

Concerning simulations, the CloudSim toolkit[17] has been widely used to propose and evaluate certain heuristics for data center consolidation, such as in References 18 and 19. The simulator used in this work, DISSECT-CF, is another publicly available open-source cloud simulator.[20] DISSECT-CF features both higher accuracy and higher simulation performance than CloudSim.[13]

Many researchers have proposed applying population-based metaheuristic algorithms to the VMCP.[10,11,21-23] Metaheuristics are nature-inspired algorithms which are widely used to solve optimization problems yielding near-optimal solutions in a reasonable amount of time. Population-based metaheuristics use a set of potential solutions and different operators to create new solutions from the already identified ones. Popular population-based metaheuristics include GA, ABC, and PSO. The first GA using crossover, recombination, mutation and selection of genes and chromosomes was invented by John Holland in the 1960s.[24] The ABC algorithm imitates a honey bee swarm.[25] The PSO algorithm also belongs to the population-based metaheuristics as it is based on the behavior of birds and fish.[26]

Table 1 gives an overview of the different VM consolidation approaches using population-based metaheuristics that have been proposed so far in the literature.

Several different problem formulations for the VMCP exist in the literature regarding, for example, the considered resources, the used techniques, and the targeted objectives. Liu et al.[28] considered CPU and memory as resources and used a weighted objective function with the primary target of reducing the number of servers and the secondary target of achieving high utilization of each PM. Farahnakian et al.[29] considered CPU and memory as resources, too, but their aim is to reduce the energy consumption by reducing the number of active PMs. The VMCP is divided into the subproblems of PM status detection and managing consolidation, which are addressed with a local controller inside each PM and a global controller for consolidation, respectively. In addition, the k-nearest neighbor heuristic and a method to predict overloaded PMs are considered. Deng et al.[35] took CPU, memory, disk utilization, and network utilization as resources into account. Their objectives were to minimize VM migration costs, to maximize data center lifetime, to minimize the amount of service level agreement (SLA) violations and to minimize the energy consumption, so there is also the focus on avoiding unprofitable and aggressive reconfigurations. Qiu et al.[36] considered CPU, RAM, and bandwidth as resources. Their objectives, for example, minimizing the number of used PMs or minimizing the number of migrations, are weighted and their algorithm used a ranking system for each individual. In addition, they implemented subalgorithms which are chosen randomly, either in favor of minimizing the number of used PMs, achieving better load balance, or reducing the communication costs of VMs. All these different problem variants make the comparison of the proposed approaches very difficult.

Beside the most popular metaheuristics mentioned previously, there are many more. Zheng et al.[22] implemented a biogeography-based optimization algorithm and Kansal and Chana[34] used a firefly optimization algorithm to solve the VMCP. Ant colony optimization was used in several articles.[23,28,29] Because of the variety of metaheuristic algorithms with similar principles but subtle details in their operation, it is difficult to determine the best one. In addition, population-based metaheuristics can be combined with local search to improve the solutions in the population, which may lead to better results overall.[38]

A challenge of the VMCP is the handling of multiple objectives, for example, the number of active PMs and the number of migrations, in the objective function. There are many different solutions for this in the literature. Li et al.[11] considered only one metric, the energy efficiency factor of each PM, to assess a given solution. Farahnakian et al.[29] used a weighted sum, attaching a higher weight to the number of active PMs and a lower weight to the number of migrations. References 23, 10, 28, and 31 implemented different combined objective functions. For example, Feller et al.[23] used a weighted product, attaching the highest weight to the number of active PMs, while Jiang et al.[10] considered an unweighted product of the total energy consumption and the SLA violation rate. Qiu et al.[36] implemented a weighted function with the possibility to favor one or more metrics which shall be minimized (called Target) while ensuring that the other objectives do not become worse than before by using them as constraints (called Keep). The best solution is then chosen based on the ordered metrics inside Target from first to last. Overall, finding the best way to handle the conflicting metrics can be challenging.

There are also differences between the proposed approaches in the way they encode potential solutions as individuals in a population. The most used way to encode solutions of these algorithms is a VM-to-PM mapping, that is, specifying for each VM the PM that should host it.[11,23,33,35,36] A slightly different way is used by Hallawi et al.:[9] each solution is represented by an array of structures, where each structure contains a host PM and an amount of VMs to be migrated from this PM. Another approach is introduced by Jiang et al.:[10] they used two queues as their solution encoding, one migrant VM queue for VM selection and a target host queue for VM allocation. References 29 and 31 used a migration plan as the encoding which contains all necessary migration tuples. These tuples contain the source PM, the VM to be migrated and the destination PM.

TABLE 1 Overview of VM consolidation approaches using population-based metaheuristics

Article	Considered resources	Algorithm	Metrics	Comparison with	Evaluation environment	Test workload	Population size	Solution encoding
	CPU RAM other		PM utilization migrations active PMs SLA violations energy consumption makespan algorithm runtime other	Simple heuristics ot her metaheuristics	Real infrastructure CloudSim other established simulator own simulation	Real workloads predefined workload random workload		VM–PM mapping migration plan other
10	●●●	ABC	●●●●●○○○	●1	○●○○	1○○	not mentioned	○○●
27	●●●	ABC	●●○○●○○○	●1	○●○○	2○○	25 bees	●○○
23	●●●	ACO	●●●○○○○○	●0	○○○●	0●○	2 ants	●○○
28	●●○	ACO	●○●○○○○○	●0	○○○●	0○●	10 ants	●○○
29	●●○	ACO	●●●●●○●○	●0	○●○○	1○●	10 ants	○●○
30	●○●	ACO	●○●○●○○●	●0	○○●○	1○○	100 ants	○○●
31	●●●	ACO	●●●●●○○●	●1	○●○○	1○●	10 ants	○●○
32	●●●	ACO	○●●●●○○○	○1	○○○●	0○●	10 ants	○●○
22	●●●	BBO	●●○○●○●●	●2	○○○●	0○●	18 individuals	●○○
33	●●○	BBO	●○●○●○●○	●2	○○○●	2○●	12 individuals	●○○
34	●●○	FFO	●●●○●○○○	●1	○●○○	0○●	not mentioned	○○●
21	●●○	GA	●○●○●○●●	●1	○○○●	2○●	not mentioned	○○●
35	●●●	GA	●●○●●○●●	●0	○●○○	0○●	not mentioned	●○○
9	●●○	GA	●○●○○○○○	●0	○○○●	0●●	75 individuals	○○●
36	●●●	GA	●●●○○○○●	●0	○●○○	1○○	100 individuals	●○○
11	●○●	PSO	●●●○●○○○	●0	○●○○	0○●	20 particles	●○○
37	●●○	PSO	●●●○●○○○	○3	○○○●	1○○	100 particles	●○○

Abbreviations: ABC, artificial bee colony; ACO, ant colony optimization; BBO, biogeography-based optimization; FFO, firefly optimization; GA, genetic algorithm; PM, physical machine; PSO, particle swarm optimization; SLA, service level agreement; VM, virtual machine.

The extensive empirical evaluation of metaheuristic-based algorithms is important because the effectiveness of metaheuristics usually cannot be proven theoretically. Reference 30 used CloudSim and Google trace data for their experiments. Reference 29 also used CloudSim as their simulator. In addition, simulated workloads and 10 days of real workload out of PlanetLab were used to test their algorithm. Reference 21 used Matlab for evaluation and considered real and simulated data for their experiments. Their simulation duration was 168 h for each considered method. In general, different simulators and workload traces are used for evaluation. In most cases, the evaluation was limited, for example, only a small number of experiments was carried out or only short traces were used, only synthetic data were used, or the evaluation was performed with in an unspecified simulation environment the accuracy of which is unknown.[9,11,22,23,28,34]

The metaheuristics proposed in the literature were often only compared with greedy heuristics, for example, first fit decreasing or BF. This is not sufficient to prove the quality of the proposed algorithms. It is also necessary to compare different metaheuristics to each other to show their individual influence on the important metrics such as resource usage or number of migrations.

Overall, the analysis of related work shows the lack of objective and unbiased comparative studies of population-based metaheuristic VM consolidation approaches using a variety of real-world workloads. Such comparisons are hampered by the many different problem variants, objective functions, and solution encodings used in the proposed approaches.

3 | OUR PROPOSED FRAMEWORK FOR DATA CENTER CONSOLIDATORS

DISSECT-CF is a compact open-source[20] simulator focusing on the internals of IaaS systems. Figure 1 presents its architecture. DISSECT-CF consists of subsystems (framed with dashed lines), each responsible for a particular functionality:

- event system: the primary time reference;
- unified resource sharing: models low-level resource bottlenecks;
- energy modeling: for the analysis of energy-usage patterns of resources (e.g., CPUs and network interfaces) and their aggregations;
- infrastructure simulation: for physical/virtual machines, sensors and networking;
- infrastructure management: provides a cloud-like API, cloud-level scheduling, and system monitoring and management.

3.1 | Foundation for consolidation algorithms

Data center consolidation techniques are heavily used in commercial clouds. Consolidation is built on the migration capabilities of VMs, where virtualized workload is moved around in the data center according to the cloud operator's goals, also taking into account workload characteristics and tenant requirements. In the past years, there were several approaches proposed for consolidating the virtualized workloads of clouds.[6] Most of them were evaluated with simulations. When analyzing cost models, the effects of consolidation could not be avoided. The foundations for these consolidator algorithms were laid down in our DISSECT-CF simulator from the beginning.[13] As a next step, more precise live migration modeling was incorporated.[39] What was missing was the implementation of actual consolidation algorithms in the simulator. The basic layout for the consolidation (including a more rudimentary version of the SimpleConsolidator discussed below) was laid out in Reference 40. In the following two paragraphs, we recite our prior art relevant for this article, then we continue discussing how we extended on the originally offered techniques and foundations to reach a more generic and approachable framework for VM consolidation research and development.

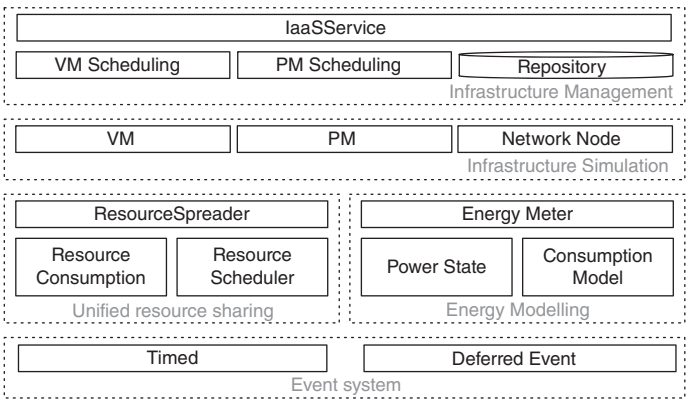

FIGURE 1 The architecture of the DISSECT-CF simulator

There are two distinct approaches possible to implement a consolidation algorithm in DISSECT-CF: (*i*) create a new PM controller which incorporates consolidation techniques into the management of PMs or (*ii*) create an independent consolidator which builds on top of the other infrastructure management components of the simulator. While both approaches could apply the same policies and enact the same goals of a cloud provider, they should be implemented differently. In the first case, the PM controller should extend its possible actions from switching on/off PMs to migrating VMs as well. In the second case, the consolidator is dedicated to only deciding on migration-related actions. This is beneficial as the consolidator algorithm could collaborate with multiple PM controller strategies without the need for a complete rewrite of the consolidation approach. The first approach offers a tighter integration of PM management and workload consolidation, but it is less flexible than the second approach with which one could pair arbitrary consolidators with arbitrary PM controllers. Therefore, in the rest of the article, we focus on the second approach.

As a generic foundation (shown on the top of Figure 2), the `Consolidator` class was introduced into the simulator. This abstract class handles the basic connection of the future consolidators to the `IaaSService`. It is also responsible for regularly invoking the custom consolidation algorithm implemented in the subclasses. For this, the minimum time interval Δt between algorithm invocations can be configured. The delay between two consecutive invocations is guaranteed to be Δt only if there are VMs hosted on the cloud; if there are no VMs, the algorithm is not invoked. To ensure this behavior, the `Consolidator` class observes how PMs are managed and utilized by PM controllers and VM schedulers. Finally, the class defines the `doConsolidation()` method to interface with custom/future consolidation algorithms which must implement this method. The `doConsolidation()` method receives the current state of the controlled cloud infrastructure as a list of PMs, that we will denote as $P := \{p_1, p_2, .., p_{i-1}, p_i, p_{i+1}, .., p_N\}$ in the rest of the article. Here, p_i denotes a particular PM, N is the number of PMs available for consolidation (at the moment of the invocation of the algorithm). Note that the time required to make a single consolidation decision is expected to be negligible compared with the total execution time of the cloud's workload. Thus, the `Consolidator` class suspends the simulation until the consolidation algorithm makes its decision.

3.2 | Model-based consolidation

Several consolidators evaluate multiple possible mappings of the VMs on the PMs before making a decision. DISSECT-CF's existing infrastructure modeling classes (shown with gray color in Figure 2) did not support such experimentation directly as they were designed to enact realistic behavior

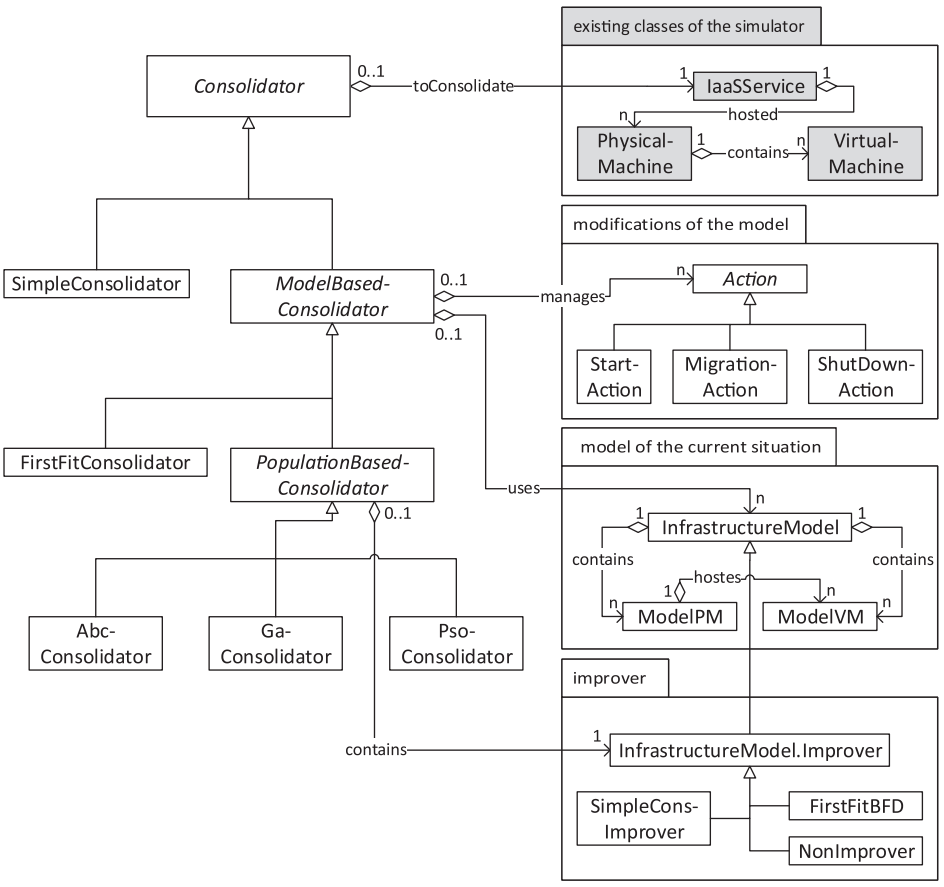

FIGURE 2 Consolidation-related extension of DISSECT-CF

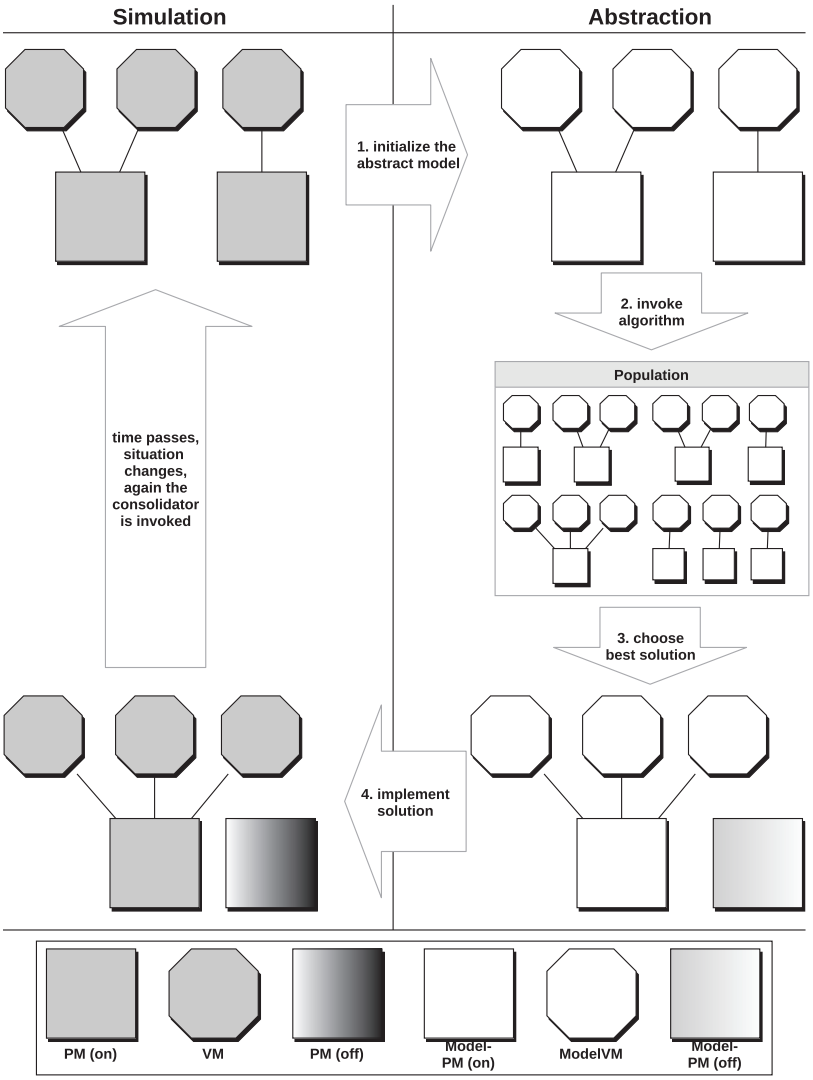

FIGURE 3 Consolidation cycle by using an abstract model of the PMs and VMs. PM, physical machine; VM, virtual machine

of real-life entities.[14] Thus, to foster the implementation of consolidators that evaluate multiple possible mappings, we extended DISSECT-CF with a more abstract model of the simulated entities (called `ModelVM` and `ModelPM`). These model objects allow multiple versions of the whole abstracted infrastructure to be present during the consolidation process (possibly at the same time, as is typical in population-based algorithms) and enable tentative changes as well as the evaluation against various metrics that drive the consolidators.

When implementing consolidators, a common pattern can be used to deal with these abstractions, as shown in Figure 3. To simplify research on future consolidation techniques, we have encapsulated this process into the `ModelBasedConsolidator` (cf. Figure 2). This class is responsible for maintaining the link between the abstract representation (i.e., the set of ModelPM and ModelVM instances, which we will denote by *MP* and *MV*, respectively) and the instances of the simulator's internal PhysicalMachine and VirtualMachine classes (which we denote by *P* and *V*, respectively). When the consolidator is invoked, the abstract model is created on the basis of the actual situation in the simulator (step 1 in Figure 3). The algorithm can freely work with the abstract model, for example, create and evolve a population of possible mappings of VMs on PMs (step 2). At the end of the algorithm, the best solution is selected (step 3) and enacted in the simulator (step 4).

Algorithm 1 shows the operation of the `ModelBasedConsolidator` from an algorithmic point of view. At first the abstract model is instantiated, followed by the invocation of the actual consolidation algorithm, which works on the abstract model and is implemented by appropriate subclasses. Finally, the solution determined by the algorithm must be enacted in the simulation. This may involve different kinds of actions, like turning PMs on or off and migrating VMs between PMs. There can also be dependencies among these actions: for example, a migration to a switched-off PM can only be executed once the target PM has been turned on. The actions, together with the dependencies, form a directed acyclic graph (DAG) and the actions must be executed in a topological order of this DAG. The creation and scheduling of the actions is performed automatically by the `ModelBasedConsolidator` (see lines 4–6 in Algorithm 1).

Algorithm 1. Algorithmic framework of the `ModelBasedConsolidator`

```
1: procedure DoConsolidation(pmList)
2:     instantiate the abstract model
3:     perform consolidation        → Invocation of the specific algorithm implemented in the subclass
4:     create all actions
5:     create the graph of dependencies among the actions
6:     execute the actions in topological order
7: end procedure
```

The `Action` class represents actions like starting a PM, stopping a PM, or migrating a VM. For the different types of actions, different subclasses are implemented, namely, `StartAction`, `ShutDownAction`, and `MigrationAction` (cf. Figure 2). To represent the complete and free resources offered by PMs (denoted by $r_c(p_i)$ and $r_f(p_i)$, respectively) we use `ResourceConstraints` objects, which already exist in the simulator.

3.3 | Foundations for metaheuristics

To ease specifically the development of metaheuristic-based consolidation algorithms, we introduce a further class called `SolutionBasedConsolidator`. This class extends the `ModelBasedConsolidator` with the functions that are useful for creating initial solutions for a subsequent heuristic search. In particular, "first fit," "random," and "unchanged" solutions are supported. A "first fit" solution is created with the first fit heuristic, a "random" solution contains a random assignment of VMs to PMs, and an "unchanged" solution mirrors the current situation of the IaaS in the simulator without any modifications.

All metaheuristic-based algorithms use the `Fitness` class to determine the fitness value of a given solution (cf. Figure 2). It consists of multiple objective function values, namely, (i) the number of overloaded PMs, (ii) the number of active PMs, and (iii) the number of migrations. Note that the fitness is not converted to a single numeric value. However, the `Fitness` class offers a function to compare two Fitness objects to each other to decide which one is better.

The `Solution` class represents a solution, which can be used as an individual in a population, and contains a mapping of VMs to PMs. Furthermore it offers various functions like initialization of the solution or determining its current fitness.

4 | CONSOLIDATION TECHNIQUES

In this section, we first describe the exact problem variant addressed by the implemented algorithms, and then the algorithms themselves, consisting of two single-solution heuristics and three population-based metaheuristics.

4.1 | Addressed problem variant

For comparing the performance of different algorithms, it is essential that they solve the same problem. As shown in Section 2, the VMCP has many variants, so we have to define the exact problem variant that is addressed by the implemented algorithms.

The problem inputs consist of (see also Table 2):

- The set of available PMs $P = \{p_1, p_2, \dots, p_N\}$. For each PM p_i, its *capacity* is given by $c(p_i) \in \mathbb{R}^3$. The three components of $c(p_i)$ correspond to (i) the number of CPU cores, (ii) the processing power per CPU core, and (iii) the memory size, respectively.
- The set of VMs $V = \{v_1, v_2, \dots, v_M\}$. For each VM v_j, the amount of its *required resources* is denoted by $r(v_j) \in \mathbb{R}^3$. The three components of $r(v_j)$ correspond to the same resource types as the ones of $c(p_i)$.
- The current *allocation* of VMs on PMs, given by $h : P \to 2^V$, where for a PM p_i, $h(p_i) \subset V$ denotes the set of VMs allocated on p_i. The $h(p_i)$ sets must build a partition of V, that is, $\cup_{p_i \in P} h(p_i) = V$ and $h(p_i) \cap h(p'_i) = \emptyset$ for any $p_i \neq p'_i$.

TABLE 2 Summary of notation

Notation	Explanation	Notation	Explanation
P	Set of all physical machines in the cloud	p_i	Physical machine i
V	Set of all virtual machines in the cloud	v_j	Virtual machine j
MP	Set of all ModelPMs in the model of the cloud	mp_i	ModelPM i
MV	Set of all ModelVMs in the model of the cloud	mv_j	ModelVM j
$h(p_i)$	Set of VMs hosted by PM i	$h(mp_i)$	Set of ModelVMs hosted by ModelPM i
$u(p_i)$	Utilization of PM i	$u(mp_i)$	Utilization of ModelPM i
$r(v_j)$	Amount of resources required by VM j	$r(mv_j)$	Amount of resources required by ModelVM j
$c(p_i)$	Capacity of PM i	$c_f(p_i)$	Available capacity of PM i
$c(mp_i)$	Capacity of ModelPM i	$c_f(mp_i)$	Available capacity of ModelPM i
$upperT$	Threshold above which the load of a PM is considered high	$lowerT$	Threshold below which the load of a PM is considered light
$popsize$	Population size	$iter$	Maximum number of iterations
$fit(k)$	Fitness value of individual k		

Based on these inputs, the *utilization* and the *available capacity* of PM p_i, denoted by $u(p_i) \in \mathbb{R}^3$ and $c_f(p_i) \in \mathbb{R}^3$, respectively, can be computed:

$$u(p_i) = \sum_{v_j \in h(p_i)} r(v_j), \quad c_f(p_i) = c(p_i) - \sum_{v_j \in h(p_i)} r(v_j). \tag{1}$$

Note that the arithmetics in (1) is between three-dimensional (3D) vectors. A PM p_i is *overloaded* if at least one component of $c_f(p_i)$ is negative. The aim of VM consolidation is to devise a new allocation $h' : P \to 2^V$ that minimizes the following three objective functions:

- The number of used PMs, where a PM p_i is *used* if $h'(p_i) \neq \emptyset$.
- The number of overloaded PMs.
- The number of migrated VMs. A VM v_j is *migrated* if $v_j \in h(p_i)$ and $v_j \in h(p'_i)$ for some PMs $p_i \neq p'_i$.

Note that VM consolidation is a multicriteria optimization problem.

4.2 | Single-solution heuristics

As a comparison baseline, we implemented two simple heuristic algorithms that constructively solve the VMCP by building a (single) solution.

4.2.1 | The simple consolidator

The simulator offers a heuristic consolidation algorithm called `SimpleConsolidator`. This algorithm aims to pack the VMs to the smallest number of PMs as shown in Algorithm 2. Notice that this consolidator is not working on an abstract model, but directly operates on DISSECT-CF's IaaS representation. The algorithm migrates the VMs from the lightest loaded PMs (identified as p_{src}) to the heaviest loaded ones (called p_{trg}), similarly as in the approach of Shi et al..[12] Only those machines are selected as heavy loaded machines that have at least a nonnegligible amount of free resources. The algorithm only moves VMs (v_j) around that are not subject to VM management operations (boot, migrate, suspend, destroy, and so forth) at the moment when the heuristic is invoked. This technique ensures that VMs are not moved around constantly in the data center by guaranteeing that a migration decision will not be reverted or changed during the enactment of the migration. This approach is efficient with the PM controller `SchedulingDependentMachines` which switches off all unused PMs once they become freed up (i.e., once all their VMs migrate away).

The algorithm iteratively takes the PM with the highest free capacity that is hosting at least one VM (line 4 in Algorithm 2), and then tries to free it up my migrating all the VMs hosted on this PM. Hence, for each VM hosted on the selected PM (line 5), a target PM is determined, which is the

Algorithm 2. SimpleConsolidator's heuristic

```
1:  procedure doConsolidation
2:      P_t ← P
3:      while P_t ≠ ∅ do
4:          p_src ← p_i ∈ P_t : |h(p_i)|>0 ∧ c_f(p_i)>0 ∧ c_f(p_i) = max_{p∈P_t} c_f(p)
5:          for ∀v_j ∈ h(p_src) do
6:              if v_j is in running state then
7:                  p_trg ← p_i ∈ P_t : |h(p_i)|>0 ∧ c_f(p_i) ≥ r(v_j) ∧ c_f(p_i) = min_{p∈P_t} c_f(p)
8:                  if p_trg ≠ null then
9:                      Initiate migration of v_j from p_src to p_trg
10:                     if c_f(p_trg) − r(v_j) is negligible then
11:                         P_t ← P_t \ {p_trg}
12:                     end if
13:                 end if
14:             end if
15:         end for
16:         P_t ← P_t \ {p_src}
17:     end while
18: end procedure
```

PM with smallest free capacity that can host the VM (line 7). If such a target PM could be found, the migration is initiated (lines 8–9). As a result, the target PM may get practically full, in which case it is removed from the list of candidate PMs (lines 10–12). This procedure is repeated for all VMs of the source PM, and then the next PM is considered as potential source.

It should be noted that the checks $c_f(p_i) > 0$ and $c_f(p_i) \geq r(v_j)$ in lines 4 and 7 must hold for each dimension of the 3D vectors. For determining the PM with maximum or minimum free capacity (lines 4 and 7), we need a total order among 3D vectors. This order is based on the product of the coordinates.

4.2.2 | First fit best fit decreasing consolidator

We also implemented a first-fit-best-fit-decreasing based algorithm to solve the VMCP. Unlike the `SimpleConsolidator`, this algorithm works on the model of the allocation, not directly on the allocation in the simulator. Thus, it is implemented as an extension of the `ModelBasedConsolidator`, and can be invoked in line 3 of Algorithm 1. The first fit best fit decreasing (FFBFD) Consolidator works with the set of model PMs *MP* and the set of model VMs *MV*.

The pseudocode of the `FFBFDConsolidator` is shown in Algorithm 3. Similarly as in the approach of Beloglazov and Buyya,[2] the aim is to completely empty lightly loaded PMs and to decrease the load of highly loaded PMs. Whether a PM is considered to be lightly loaded or highly loaded is decided based on given lower and upper thresholds *lowerT* and *upperT*, where $0 \leq lowerT \leq upperT \leq 1$.

The algorithm iterates at first over the set of model PMs. If the next model PM mp_i hosts VMs and is either lightly loaded or highly loaded (lines 3–4 of Algorithm 3), then we move VMs hosted on mp_i to a list until the load either vanishes or the model PM is not highly loaded anymore (lines 5–6). The list of VMs thus collected and a copy of *MP* are sorted according to the BF decreasing order afterward (lines 9–10). Finally, each model VM mv_i in the list is migrated to the first host mp_j that can host it under the condition $c_f(mp_j) \geq r(mv_i)$; if no suitable host is found, mv_i is put back on its previous host (lines 11–18).

While the basic idea of the FFBFD Consolidator is similar to that of the Simple Consolidator, there are three major differences. First, the Simple Consolidator does not aim for reducing the load on highly loaded PMs, whereas this is done by the FFBFD Consolidator as a precaution to avoid PM overloads. Second, the FFBFD Consolidator only tries to empty the lightly loaded PMs, whereas the Simple Consolidator investigates potentially significantly more PMs that could be emptied. Third, the FFBFD Consolidator only performs the migrations from a lightly loaded PM if all affected VMs can be migrated, whereas the Simple Consolidator performs the migrations even if only a subset of the VMs can be migrated.

The latter point is also the reason why a model-based approach is advantageous for the FFBFD Consolidator: the migrations can be performed tentatively on the model and can even be undone if necessary, without incurring the costs of real migrations. Real migrations are only carried out when the solution has been determined.

Algorithm 3. First fit best fit decreasing consolidation

1: **procedure** OPTIMIZE
2: initialize $MV_{toMigrate}$ as an empty set of ModelVMs.
3: **for** $mp_i \in MP$ **do**
4: **while** $h(mp_i)>0$ and $(u(mp_i) \leq c(mp_i) \cdot lowerT$ or $u(mp_i) \geq c(mp_i) \cdot upperT)$ **do**
5: add the last mv_{last} to $MV_{toMigrate}$
6: remove mv_{last} of mp_i
7: **end while**
8: **end for**
9: sort $MV_{toMigrate}$ according to BFD
10: initialize MP_{sorted} as a sorted clone of MP according to BFD
11: **for** $mv_i \in MV_{toMigrate}$ **do**
12: find the first $mp_j \in MP_{sorted}$ to host mv_i under the condition $c_f(mp_j) \geq r(mv_i)$
13: **if** a new host has been found for mv_i **then**
14: migrate mv_i to its new host
15: **else**
16: put mv_i back on its previous host
17: **end if**
18: **end for**
19: **end procedure**

4.3 | Population-based metaheuristics

This section describes the implementation of the population-based metaheuristic consolidation algorithms. As shown in Section 2, several population-based metaheuristics have been suggested to solve the VMCP. We decided to implement three different and popular metaheuristics, namely, GA (for reference see 9, 36), ABC (for reference see 10), and PSO (for reference see 11). All these algorithms use a *population* (sometimes also called swarm) consisting of *individuals* where each individual encodes a possible solution of the VMCP. Different *operators* are used to alter existing individuals or create new individuals. A *fitness* function is used to evaluate the merits of individuals.

To enable a meaningful comparison, we implemented the three algorithms in a consistent manner, as much as this was possible given their algorithmic differences. In particular, we used the same encoding of individuals, the same fitness function, and the same algorithmic framework for handling the population, the same procedure for generating the initial population, and the same local search procedures to improve individuals.

The encoding of individuals is defined in the `InfrastructureModel` class (cf. Figure 2). A solution to the VMCP is stored in the form of a VM-to-PM mapping, that is, for each ModelVM, the ModelPM that should host it is stored.

The fitness function assigns to each individual three numbers: the total of ModelPM overloads, the number of used ModelPMs, and the number of migrations. This directly mirrors the three objectives of the VMCP as defined in Section 4.1. Since the fitness is vector-valued, this makes it nontrivial to compare the fitness of two individuals. For this purpose, we use lexicographic ordering. That is, we first compare the two individuals in terms of PM overloads, and if one individual is better than the other in this respect by at least 1%, then its fitness is considered better than that of the other individual. If the two individuals lead to practically the same total of PM overloads, then they are compared in terms of the number of used PMs. If one of them is better in this respect, then its fitness is considered better. If the two individuals are equal also in this regard, then the number of migrations decides. Using this lexicographic ordering, minimizing the total of PM overloads is the primary objective, minimizing the number of used PMs is the secondary objective, and minimizing the number of migrations is the tertiary objective. The notation $fit(x) > fit(y)$ is used to denote that the fitness of individual x is better than the fitness of individual y. The fitness comparison logic is also implemented in the `InfrastructureModel` class.

Algorithm 4 shows the general algorithmic framework for population-based metaheuristics. The algorithms start by creating the initial population (line 2). To achieve both diversity and quality, we use multiple methods for generating individuals for the initial population. First, we add some individuals corresponding to the current placement of VMs on PMs in the simulator. Second, a given number of individuals created with the First Fit Consolidator are added. Third, the rest of the population is filled with individuals that are created randomly and then improved using first fit. The creation of the initial population may be followed by some algorithm-specific initialization steps (line 3), and a given number of iterations of steps (lines 4–10) altering the population, which are again algorithm-specific (line 6). If there is no improvement in the population in iteration, the algorithm is stopped to avoid wasting further computation time. Finally, the best solution found is transferred to the simulator (line 11).

Algorithm 4. Framework for population-based metaheuristics

```
1:  procedure OPTIMIZE
2:      create initial population
3:      init()          → algorithm-specific initialization steps
4:      for i = 1, … , iter do
5:          improved = false
6:          oneIteration()          → algorithm-specific steps to alter the population
7:          if improved != true then
8:              break
9:          end if
10:     end for
11:     implement the found best solution in the simulator
12: end procedure
```

To boost the performance of population-based metaheuristics, they can be combined with local search to improve the solutions in the population, which may lead to better results overall. This optimization technique is well known in the evolutionary computation community.[41] In VM consolidation, most existing approaches using population-based metaheuristics do not use local search, but some do.[21,38] As the application of these local searches could interfere with the applied metaheuristics, when a new simulation is initiated these can be configured to be inactive. This configuration option allows direct observation of the original metaheuristics and allows direct comparison with results found in other works. If a new metaheuristic is devised this configuration also allows to check if local search has positive or negative impact on its performance.

We implemented two local search procedures to optionally improve particular individuals in the population. Our two procedures correspond to the algorithms of the SimpleConsolidator (implemented by the `SimpleConsImprover` class in Figure 2) and the FirstFitConsolidator (implemented by `FirstFitBFD`), respectively. Any of the implemented population-based metaheuristics can simply invoke a method called `improve()` to improve a modified or newly created individual; the method will execute one of the local search procedures, depending on a parameter setting.

4.3.1 | Genetic algorithm

The pseudocode of our GA-based algorithm is shown in Algorithm 5. There are no specific instructions needed before starting the consolidation loop, so the `init()` method is not required.

For each individual (parent) in the population we create a new solution (child) by mutation (line 3). Note that the parent is not changed. To fill the VM-to-PM mapping for the child, we use the following procedure: for each VM, with probability *mutationProb* a random PM is selected, and with probability 1 – *mutationProb* the same PM as in the parent. Afterward local search is used to improve the child (line 4). If the child has a better fitness than the parent, then the parent gets replaced by the child in the population (lines 5–8).

After the mutations, a given number (*nrCrossovers*) of crossovers is performed. To do a crossover, we pick two random individuals (r1, r2) from the population and recombine them to a new individual r3 which we then improve using local search (lines 11–13). If the fitness of the newly created offspring is better than one of the two parents, then this parent is replaced by the offspring in the population (lines 14–20).

If there was any improvement in the population—either as the result of a mutation or as the result of a recombination—then improved is set to true (lines 7, 16, 19), otherwise it remains false. This information is used in Algorithm 4 to determine if the algorithm should be continued with the next iteration.

4.3.2 | Artificial bee colony

The ABC algorithm is inspired by the behavior of a bee colony, in which bees with different roles cooperate to find the best food source.[25] Our ABC implementation is based on the pseudocode given by Mernik et al.,[42] but adopts it to our problem model and algorithmic framework.

The pseudocode of our ABC-based algorithm is shown in Algorithms 6 and 7. For an individual x in the population, trials(x) denotes the number of times the algorithm has tried to improve x without success since the last successful improvement. In the `init()` method, trials(x) is initialized

Algorithm 5. Genetic algorithm

```
1:  procedure ONEITERATION
2:      for parent ∈ population do
3:          child = mutate(parent)
4:          improve(child)
5:          if fit(child) ≻ fit(parent) then
6:              replace parent with child in the population
7:              improved = true
8:          end if
9:      end for
10:     for n = 0; n < nrCrossovers; n++ do
11:         r1, r2 = two random individuals in the population
12:         r3 = crossover(r1, r2)
13:         improve(r3)
14:         if fit(r3) ≻ fit(r1) then
15:             replace r1 with r3 in the population
16:             improved = true
17:         else if fit(r3) ≻ fit(r2) then
18:             replace r2 with r3 in the population
19:             improved = true
20:         end if
21:     end for
22: end procedure
```

Algorithm 6. Artificial bee colony—Part 1

```
1:  procedure INIT
2:      for x ∈ population do
3:          trials(x) ← 0
4:      end for
5:  end procedure
6:  procedure ONEBEE(x : an individual in the population)
7:      x' = mutate(x)
8:      improve(x')
9:      if fit(x') ≻ fit(x) then
10:         replace x with x' in the population
11:         trials(x)=0
12:         improved = true
13:     else
14:         trials(x)++
15:     end if
16: end procedure
```

to be 0 for each individual (Algorithm 6, lines 2–4). Later on, when the algorithm tries to improve x without success (Algorithm 6, line 14), trials(x) is increased. When a successful improvement takes place (Algorithm 6, line 11) or the individual is replaced by a freshly created one (Algorithm 7, line 26), trials(x) is reset to 0.

The bee colony consists of three groups of bees: employed bees, onlooker bees, and scout bees, where each type of bee has its own acting phase within the `oneIteration()` method.

The employed bees phase (Algorithm 7, lines 3–5) and the onlooker bees phase (Algorithm 7, lines 7–19) consist of a set of `oneBee()` calls. Each `oneBee()` call tries to improve an individual by a mutation and a subsequent local search; the improvement is successful if it leads to better

Algorithm 7. Artificial bee colony—Part 2

```
1:  procedure ONEITERATION
2:      //employed bees phase
3:      for x ∈ population do
4:          ONEBEE(x)
5:      end for
6:      //onlooker bees phase
7:      for x ∈ population do
8:          sample ← set of κ randomly chosen individuals of the population
9:          p(x) ← |{y ∈ sample : fit(x) > fit(y)}|/κ
10:     end for
11:     j = t = 0;
12:     while t<popsize do
13:         r = random double value from [0,1]
14:         if r<p(j) then
15:             ONEBEE(j)
16:             t++
17:         end if
18:         j ← ((j + 1) mod popsize)
19:     end while
20:     //scout bee phase
21:     n = individual with highest trials
22:     if trials(n) ≥ limitTrials then
23:         create a new random individual s
24:         improve(s)
25:         replace n with s in the population
26:         trials(s) ← 0
27:     end if
28: end procedure
```

fitness, otherwise it is discarded (Algorithm 6, lines 6–16). The difference between the employed bees phase and the onlooker bees phase is that in the employed bees phase `oneBee()` is called for each individual exactly once, whereas in the onlooker bees phase individuals with better fitness are preferred. In the original ABC algorithm, this is achieved by computing probabilities for the individuals that are proportional to their fitness. In our case, since the fitness of an individual is not a single number, a different approach was needed. To estimate how good an individual x is compared with its peers, we randomly sample κ other individuals from the population (where κ is a given positive integer) and compare the fitness of x to each of them (Algorithm 7, lines 7–10). The percentage of the individuals that are worse than x in the sample is taken as the probability p(x), and `oneBee(x)` is called with that probability (Algorithm 7, lines 13–15).

Finally in the scout bee phase, if the individual with highest trials(x) has been tried to improve without success at least limitTrials times (which is a given constant), then it is replaced by a new individual (Algorithm 7, lines 21–27).

4.3.3 | Particle swarm optimization

By contrast to the GA and ABC algorithms, here the individuals (called particles in the PSO terminology) encode the VM-to-PM mapping numerically. That is, VMs are numbered from 1 to *M*, PMs are numbered from 1 to *N*, and the VM-to-PM mapping is given as an *M*-dimensional vector, each coordinate of which is in $\{1, 2, \ldots, N\}$. This vector is called the *location* of the particle. Beside its location, each particle also has a *velocity*, which is also an *M*-dimensional vector. Encoding locations and velocities in $\mathbb{R}^M$ makes it possible to perform vector arithmetics with these vectors.

For each particle *p*, the best location (in terms of fitness) it has found so far is stored in pBest(p). Moreover, the best location that any particle has found so far is stored in gBest. In each iteration, the velocity of each particle is recomputed based on its current velocity (called inertia component),

Algorithm 8. Particle swarm optimization

```
1:  procedure INIT
2:      for p ∈ population do
3:          set pBest(p) to the current location of p
4:      end for
5:      Particle bestParticle = Particle with the best fitness
6:      set gBest to the location of bestParticle
7:  end procedure
8:  procedure ONEITERATION
9:      for p ∈ population do
10:         if fit(p) ≻ fit(pBest(p)) then
11:             set pBest(p) to the current location of p
12:         end if
13:     end for
14:     Particle bestParticle = Particle with the best fitness
15:     if fit(bestParticle) ≻ fit(gBest) then
16:         set gBest to the location of bestParticle
17:     end if
18:     for p ∈ population do
19:         calculate velocity(p) based on Equations∼((2))-((6))
20:         location(p) = location(p) + velocity(p)
21:         round location(p) to integer coordinates between 1 and N
22:         improve(p)
23:     end for
24: end procedure
```

its location relative to its pBest location (cognitive component), and its location relative to the gBest location (social component), also using given and random weights. Specifically, the velocity of particle p is recomputed as follows (w, c_1, c_2 are given constants):

$$r1, r2 = \text{two random values between 0 and 1}, \tag{2}$$

$$inertiaComp = velocity(p) * w, \tag{3}$$

$$cognitiveComp = (pBest(p) - location(p)) * c1 * r1, \tag{4}$$

$$socialComp = (gBest - location(p)) * c2 * r2, \tag{5}$$

$$velocity(p) = inertiaComp + cognitiveComp + socialComp. \tag{6}$$

Note that Equations (3)–(6) use M-dimensional vector arithmetics. When $velocity(p)$ has been recomputed, also $location(p)$ can be updated using $location(p) = location(p) + velocity(p)$.

The pseudocode of our PSO-based algorithm is shown in Algorithm 8. The `init()` method is used to initialize the pBest locations of the particles (lines 2–4) and the gBest location (lines 5–6). Similarly in the `oneIteration()` method, the pBest locations of the particles are updated (lines 9–13) as well as the gBest location (lines 14–17). Then, the velocity and location of each particle is recomputed using the formulas given above (lines 19–20). Since this may lead to location coordinates outside $\{1, 2, \ldots, N\}$ which would be meaningless in our solution encoding, the coordinates are rounded (line 21), also ensuring that values lower than 1 are changed to 1 and values larger than N are changed to N. Finally, improvement using local search is performed (line 22).

5 | EVALUATION

During our implementation and evaluation, where applicable, we used publicly available information from real data center environments to populate our experiments, so as to maximize the relevance of our results to the real world. In particular, the experiments were based on 14 different real-world workload traces.

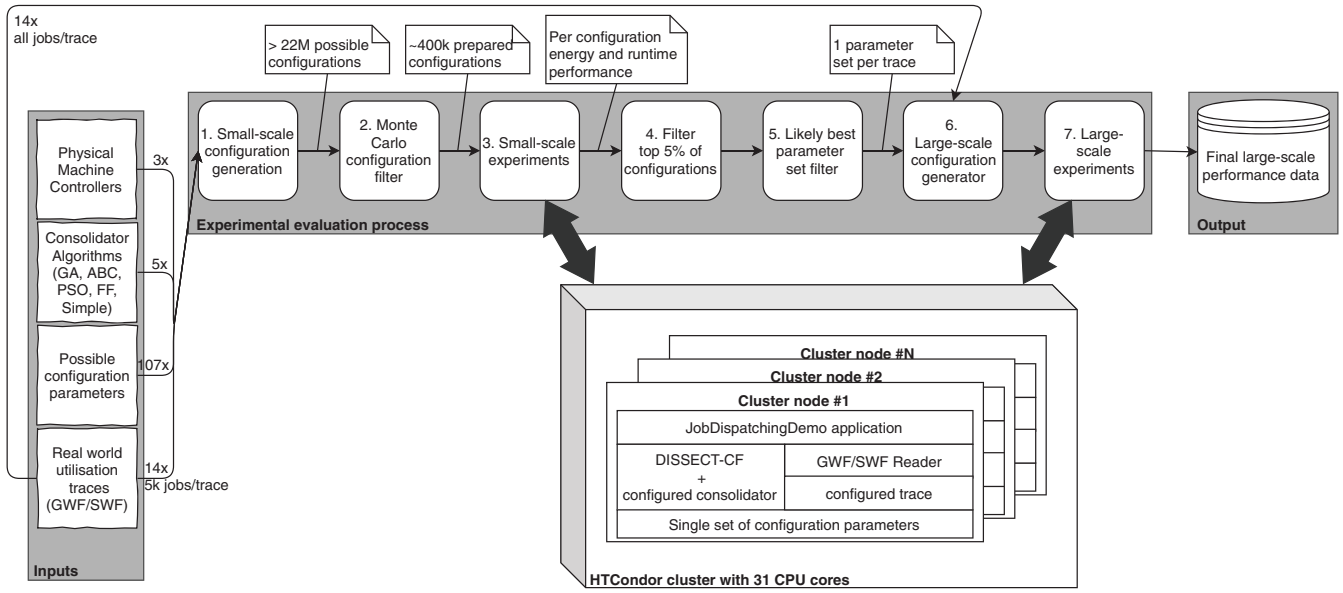

FIGURE 4 Overview of the evaluation process

A big challenge in the experimental evaluation of the implemented consolidation algorithms was the huge configuration space, resulting from the many parametrization options of the considered algorithms. Specifically, there were more than 22 million configurations to test. In addition, evaluating a configuration meant running a simulation with all traces, and since the traces are quite large, a single simulation run lasted up to several hours. To cope with this problem, we needed a more sophisticated methodology than the brute-force approach of running all configurations on all workload traces. The main idea of our evaluation methodology can be summarized as follows:

1. For each algorithm, we randomly selected a potential configuration from the previously discussed 22 million. The selection technique was done with the help of a uniform pseudorandom generator following a Monte Carlo method.[43] This allowed us to evenly sample the large search space with just 400 thousand configurations and make predictions on the rest of the configurations without exhaustively needing to test all of them.
2. For each algorithm and each trace, the selected parameter configurations were tested on the first 5000 jobs of the trace ("small-scale experiments").
3. Based on the results of the small-scale experiments, appropriate values were determined and fixed for each parameter of each algorithm, for each trace.
4. The algorithms were compared with each other on each full trace, using the parameter configurations determined previously ("large-scale experiments").

Even with this approach, the evaluation process took several months, using a HTCondor cluster with 31 CPU cores.

A more detailed overview of the evaluation process is given in Figure 4. The next subsections detail the inputs, the steps, and the results of the evaluation process.

5.1 | Inputs

We tested all five implemented consolidation algorithms (SimpleConsolidator, FFBFD, GA, ABC, PSO) described in Section 4. Each consolidator can be used in combination with one of three different PM schedulers: either one of the two built-in PM schedulers of DISSECT-CF (AlwaysOnMachines, SchedulingDependentMachines) or the newly created ConsolidationFriendly (CF) PM scheduler, which can be controlled by the consolidator directly. Each algorithm can be configured with several parameters (see also the following subsection).

Though VM management log-based traces would be the best candidates for analyzing cloud characteristics, traces collected from other large-scale infrastructures like grids are also appropriate. Generally, two main sources are used for this purpose: the grid workloads archive (GWA[1])

[1]http://gwa.ewi.tudelft.nl

TABLE 3 Values of the parameters used in the small-scale experiments

Parameter	Values
mutationProb	0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 1.00
iterations	3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 41, 61, 81
population	3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 41, 61, 81
doLocalSearch1	true, false
doLocalSearch2	true, false
limitTrials	1, 4, 5, 7, 10, 13
nrCrossovers	3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 41, 61, 81
c1	0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 1.60, 1.70, 1.80, 1.90, 2.00, 2.10, 2.20, 2.30, 2.40, 2.50, 2.60, 2.70, 2.80
c2	0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 1.60, 1.70, 1.80, 1.90, 2.00, 2.10, 2.20, 2.30, 2.40, 2.50, 2.60, 2.70, 2.80
lowerThreshold	0.20, 0.40, 0.60
upperThreshold	1.00

and the parallel workloads archive (PWA[2]). For this study we used traces downloadable from GWA (namely, AuverGrid, DAS2, Grid5000, LCG, NorduGrid, and SharcNet) as well as from PWA (IntelA, IntelB, IntelC, IntelD, LLNL-TH, METACTR2, PIK, RICC).

We used the `JobDispatchingDemo` class from the DISSECT-CF examples project[3], to transform the jobs listed in the trace to VM requests and VM activities. This dispatcher asks the simulator to fire an event every time when the loaded trace prescribes the arrival of a new job. In addition, the dispatcher maintains a list of VMs available to serve job related activities (e.g., input & output data transfers, CPU and memory resource use). Initially the VM list is empty. A job arrival event is handled with two approaches: (i) if there is no unused VM in the VM list that has sufficient resources for the prescribed job, then the dispatcher creates a VM according to the resource requirements of the job; alternatively, (ii) if there is an unused VM with sufficient resources for the job, then the job is just assigned to the VM. In the first approach, the job's execution is delayed until its corresponding VM is spawned. In both cases, when the job finishes, it marks the VM as unused. This step allows other jobs to reuse VMs pooled in the VM list. Finally, the VMs are not kept for indefinite periods of time, instead they are kept in accordance with the billing period applied by the cloud provider. This ensures that the VMs are held for as long as they were paid for but not any longer. If there is no suitable job coming for a VM within its billing period, then the VM is terminated and it is also removed from the VM list.

5.2 | Steps 1–3: Preparing and conducting the small-scale experiments

To determine the best parameter configuration for each consolidator and each trace, we defined a variety of small-scale tests with different combinations of the relevant parameters. In detail, Table 3 contains the used values for each parameter. Note that for parameters that are relevant for multiple consolidators (like "mutationProb" or "iterations"), all values are tested for each consolidator, respectively. In addition, there are two global parameters called "lowerThreshold" and "upperThreshold." Those values determine the threshold of a PM's load to determine whether it is too low or too high, resulting in either emptying the PM or moving VMs to other PMs until the overload is resolved.

From the over 22 million possible parameter configurations, we randomly selected about 400,000 configurations using Monte Carlo sampling. Testing so many configurations gave us sufficient insight into the impact of the parameter values, while still keeping the required time for the experiments manageable. Randomly selecting the configurations to test helped to avoid bias stemming from specific combinations of parameter values.

Technically, each test case is defined by an XML file containing the necessary parameters for the tested consolidator. To generate those files automatically, we implemented a class called `ConsolidationController`, that offers the possibility to create one test file for each combination of the values of the parameters.

The experiments are controlled by a separate script, which uses the `JobDispatchingDemo` to run all determined test cases for each consolidator, combined with each PM scheduler, on each trace.[20] For the small-scale experiments, only a small fraction of each trace, namely, the first 5000 jobs, were used.

[2]http://www.cs.huji.ac.il/labs/parallel/workload
[3]https://github.com/kecskemeti/dissect-cf-examples

5.3 | Steps 4–7: Preparing and conducting the large-scale experiments

The small-scale experiments delivered a large set of data, containing several metrics for each tested parameter configuration of each algorithm, combined with each PM scheduler[4], on each trace. Our next task was to identify the "best" parameter configuration of each consolidator—PM scheduler pair on each trace. This was challenging because there are several important metrics for evaluating a configuration, and the best configuration with respect to one metric may perform poorly with respect to some other metrics. In particular, we considered the following three metrics for the comparison of the different parameter configurations:

- Total energy consumption
- Duration (in real time) of the simulation
- Duration (in simulated time) of the execution of the jobs

First, we identified the best 5% parameter configurations for each trace and each consolidator—PM scheduler pair. For this purpose, we considered the above three metrics in the given order of priority, that is, energy consumption had the highest importance. We used two different filtering techniques: one based on strict lexicographic ordering of the three metrics according to their given order, and a less strict approach which allows less energy efficient configurations to be selected if their runtime is significantly better.

Next, we identified the best value for each parameter based on the values' occurrences in the top 5% of configurations. For this, we again used two different techniques. On the one hand, we selected the most frequently occurring value of a parameter in the top 5% of configurations; in the case of a tie, we selected the value that was used for achieving the best energy consumption. On the other hand, we performed a Bayesian analysis to compute the most likely best value based on the given sample of top configurations.

In addition to these techniques, for numeric parameters we also computed the Pearson correlation coefficient between the parameter and the considered metrics. In cases where a clear correlation could be established, we verified that the selected value is in line with this. For example, if there is a positive correlation between a parameter and energy consumption, then a low value should be selected for the given parameter. Finally, in cases where the values suggested by the different methods were significantly different, we used visual analysis of 3D plots (with the dimensions corresponding to the considered metrics) created with an R program to identify the reasons for the discrepancy and to decide which value to choose.

The best parameter configurations selected for each consolidator, each PM scheduler, and each trace are shown in Table 4. These configurations already lead to some interesting observations:

- It indeed makes sense to train each consolidator separately on each trace. In other words, there are no "globally good" values for the parameters. For example, the best mutation probability of the GA is 0.1 for some traces and 0.8 for some others.
- Concerning the number of iterations and the population size, it would be plausible to expect that higher values are better. However, this is often not true. In several cases, quite low values (e.g., 3) proved best. This means that also higher values did not lead to lower energy consumption, only to higher simulation duration (because of the increased algorithm execution time).
- Local search is useful. However, there is no clear winner between the two used local search procedures.

Finally, we performed the large-scale experiments. That is, we ran each consolidator, in conjunction with the CF and SDM PM schedulers, on each trace, with the parameter configuration shown in Table 4. By contrast to the small-scale experiments where only the first 5000 jobs were used from each trace, this time the full length of the traces was used (which is orders of magnitudes higher).

5.4 | Results of the large-scale experiment

Table 5 contains the results of the large-scale experiment for each used combination of consolidators and schedulers for each trace. For each test run, three metrics are shown: the total energy consumption, the duration of the simulation (in real time), and the number of migrations. (We also collected other metrics that we do not report here because they did not lead to significant insights. For example, the duration of the execution of the workloads in simulated time was also captured, but there were hardly any differences between the consolidators in this regard.) For each trace and each metric (i.e., for each row of the table) the best value achieved by any consolidator is italic in Table 5, as well as any other values that are at most 1% higher than this best value.

[4]Since the main objective of VM consolidation is to minimize energy consumption by switching off unused PMs, we did not include the AlwaysOnMachines PM scheduler of DISSECT-CF which never turns off PMs. Hence the comparison was limited to the SchedulingDependentMachines (SDM) and CF PM schedulers.

TABLE 4 Used values for each parameter

consolidator	scheduler	parameter	auver	das2	grid5000	IntelA	IntelB	IntelC	IntelD	lcg	LLNLTH	METACTR2	nordugrid	PIK	RICC	sharcnet
ABC	CF	mutationProb	0.80	0.10	0.50	0.90	0.90	0.80	0.60	1	0.80	1	0.20	0.10	0.40	0.90
		iterations	9	3	3	3	3	3	15	15	19	7	9	3	15	11
		population	3	3	3	3	3	3	3	3	3	17	3	17	3	3
		doLocalSearch1	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
		doLocalSearch2	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE
		limitTrials	13	7	10	13	10	10	10	10	10	7	10	10	13	7
ABC	SDM	mutationProb	0.20	0.90	0.80	0.20	0.10	0.70	0.10	0.10	0.90	0.20	0.70	0.10	1	0.80
		iterations	3	11	1	1	15	1	1	13	17	3	11	11	15	1
		population	3	3	3	3	13	3	13	5	1	3	1	19	3	3
		doLocalSearch1	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE
		doLocalSearch2	FALSE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
		limitTrials	10	1	10	4	13	10	13	10	10	10	10	13	13	10
GA	CF	mutationProb	0.10	0.70	0.10	0.10	0.20	0.10	0.10	0.10	0.70	1	0.80	0.30	0.70	0.70
		iterations	17	5	13	41	21	41	3	21	5	41	3	3	5	5
		population	3	3	3	3	3	3	41	3	3	3	3	3	3	3
		doLocalSearch1	false	false	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
		doLocalSearch2	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	false	TRUE	TRUE	TRUE
		nrCrossovers	3	3	13	3	3	9	3	3	3	21	5	61	3	3
GA	SDM	mutationProb	0.50	0.20	0.20	0.60	0.70	0.70	1	0.20	0.20	0.70	0.80	0.30	0.60	0.80
		iterations	1	11	11	41	7	81	3	17	21	7	11	41	3	1
		population	3	3	3	5	3	3	5	3	3	3	3	3	3	3
		doLocalSearch1	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
		doLocalSearch2	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE
		nrCrossovers	1	1	3	11	5	5	5	1	3	7	1	11	5	1

(Continues)

TABLE 4 (Continued)

consolidator	scheduler	parameter	auver	das2	grid5000	IntelA	IntelB	IntelC	IntelD	lcg	LLNLTH	METACTR2	nordugrid	PIK	RICC	sharcnet
PSO	CF	c1	1.70	2.40	0.40	0.40	0.15	0.30	1.80	0.25	2.20	0.20	0.30	0.05	0.05	0.35
		c2	0.20	2.8	0.20	0.25	0.10	2.50	2.50	0.15	0.35	0.10	0.15	0.05	0.05	0.10
		iterations	3	41	61	3	3	3	3	3	11	11	3	3	3	3
		population	3	41	3	3	3	3	7	3	3	21	11	41	5	13
		doLocalSearch1	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE
		doLocalSearch2	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
PSO	SDM	c1	2.2	0.15	1.80	2.30	0.35	2	1.80	0.30	1.70	2.30	0.40	1.70	0.05	2.60
		c2	0.35	2	1.70	0.05	0.10	0.40	0.10	0.40	0.10	0.05	0.20	0.40	0.30	0.30
		iterations	3	11	21	3	3	5	3	3	3	5	3	3	5	5
		population	7	3	41	3	3	3	3	3	21	13	11	81	61	17
		doLocalSearch1	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
		doLocalSearch2	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

Abbreviations: ABC, artificial bee colony; GA, genetic algorithm; PSO, particle swarm optimization; SDM, SchedulingDependentMachines.

TABLE 5 Results of the large-scale experiment

trace	metric	ABC-CF	ABC-SDM	GA-CF	GA-SDM	PSO-CF	PSO-SDM	SC-SDM	FFBFD-CF
auver	consumed energy (kwh)	***242,601***	246,593	***242,600***	245,495	***243,704***	***243,825***	***242,541***	247,321
auver	duration of simulation (ms)	790,696	749,127	898,250	747,485	875,909	849,414	***690,603***	732,158
auver	number of migrations	72,773	233,205	72,750	288,910	411,096	581,933	124,039	***9448***
das2	consumed energy (kwh)	***295,786***	***294,739***	***295,799***	***294,499***	***296,761***	***294,838***	***294,492***	***296,082***
das2	duration of simulation (ms)	1,482,803	***1,142,648***	1,531,607	1,206,083	5,233,036	***1,138,726***	***1,149,036***	1,172,866
das2	number of migrations	61,998	59,573	66,157	42,134	141,832	18,936	53,581	***9136***
grid5000	consumed energy (kwh)	***578,471***	***578,543***	***577,674***	***575,673***	603,196	583,752	***576,056***	583,081
grid5000	duration of simulation (ms)	2,022,123	1,623,497	2,553,549	1,771,606	3,836,820	1,640,997	***1,526,295***	1,865,579
grid5000	number of migrations	169,163	375,431	133,809	139,707	2,051,322	14,058	98,572	***9082***
IntelA	consumed energy (kwh)	***48,896***	***48,737***	***48,897***	***48,736***	49,677	49,397	***48,737***	52,260
IntelA	duration of simulation (ms)	159,922	140,553	152,225	251,389	424,148	402,119	139,977	***117,456***
IntelA	number of migrations	119,964	92,720	119,694	96,311	317,264	257,605	94,271	***5578***
IntelB	consumed energy (kwh)	***65,077***	67,934	***65,068***	***64,805***	66,128	65,687	***64,803***	69,925
IntelB	duration of simulation (ms)	152,087	8,025,616	175,628	178,636	667,375	541,144	144,784	***140,902***
IntelB	number of migrations	124,472	1,734,237	121,817	69,216	419,227	322,330	69,401	***7040***
IntelC	consumed energy (kwh)	***76,733***	***76,541***	***76,732***	***76,541***	78,740	77,688	***76,537***	86,739
IntelC	duration of simulation (ms)	165,523	200,046	341,015	243,887	1,053,842	968,301	165,231	***125,328***
IntelC	number of migrations	113,680	74,363	112,788	74,902	734,637	471,671	74,588	***4426***
IntelD	consumed energy (kwh)	***166,806***	***166,799***	***166,806***	***166,799***	***167,460***	***167,132***	***166,794***	171,149
IntelD	duration of simulation (ms)	289,034	637,491	1,426,950	695,826	5,907,145	1,310,731	***204,052***	239,706
IntelD	number of migrations	24,802	21,362	24,802	21,362	198,723	113,928	21,846	***3016***
lcg	consumed energy (kwh)	***21,417***	22,602	***21,417***	***21,410***	21,819	22,196	***21,408***	23,371
lcg	duration of simulation (ms)	55,093	185,421	46,143	47,320	73,235	95,387	37,007	***36,059***
lcg	number of migrations	29,662	404,610	29,662	28,668	134,237	282,562	31,085	***2099***
LLNL-TH	consumed energy (kwh)	1,041,626	***436,751***	1,041,626	***436,220***	1,041,626	***436,140***	***436,140***	***437,244***
LLNL-TH	duration of simulation (ms)	326,104	375,744	***321,753***	475,414	326,855	354,160	354,160	341,837
LLNL-TH	number of migrations	8030	***0***	8030	6959	8030	5727	5727	***0***
METACTR2	consumed energy (kwh)	***1,473,549***	1,511,858	***1,471,441***	***1,468,925***	1,566,826	1,509,874	***1,471,113***	1,545,138
METACTR2	duration of simulation (ms)	7,425,433	2,371,684	2,974,369	1,907,065	34,103,890	9,325,688	1,123,653	***1,017,380***
METACTR2	number of migrations	462,926	6,406,271	520,516	482,238	9,978,470	6,360,341	444,690	***18,084***
nordugrid	consumed energy (kwh)	1,140,027	1,181,924	1,129,382	1,102,303	1,137,959	1,089,088	***1,077,688***	1,102,551
nordugrid	duration of simulation (ms)	4,498,178	2,088,051	18,351,988	3,997,516	11,640,415	5,631,482	***1,715,608***	1,950,208
nordugrid	number of migrations	13,208,637	91,154	19,824,005	7,345,082	18,950,089	4,767,739	337,083	***71,619***
PIK	consumed energy (kwh)	1,542,534	***1,352,489***	***1,344,456***	***1,340,512***	1,395,669	***1,351,689***	***1,341,174***	1,370,319
PIK	duration of simulation (ms)	6,883,569	7,265,936	7,129,424	3,393,438	11,676,174	11,743,036	***2,474,989***	2,564,651
PIK	number of migrations	21,521,828	4,324,181	258,590	226,589	9,684,692	3,356,802	242,755	***22,759***
RICC	consumed energy (kwh)	***850,820***	***851,725***	***850,691***	***849,735***	877,788	***852,862***	***849,814***	***851,458***
RICC	duration of simulation (ms)	461,519	1,288,276	594,482	621,392	2,061,391	20,195,037	409,210	***391,213***
RICC	number of migrations	109,459	220,217	109,813	106,597	2,578,095	838,694	109,343	***82,876***
sharcnet	consumed energy (kwh)	***1,167,140***	***1,165,966***	***1,167,025***	***1,165,834***	1,232,665	1,183,038	***1,166,109***	1,199,625
sharcnet	duration of simulation (ms)	1,202,105	922,815	1,519,642	1,505,811	12,085,140	17,415,840	***794,810***	847,387
sharcnet	number of migrations	414,824	380,356	422,474	382,815	13,341,063	4,502,810	394,903	***46,492***

Abbreviations: ABC, artificial bee colony; GA, genetic algorithm; PSO, particle swarm optimization; SDM, SchedulingDependentMachines.

In general, we can establish that there is *no clear winner* among the considered algorithms. That is, the best algorithm for one trace may be beaten by other algorithms on other traces regarding the same metric or on the same trace regarding other metrics. Nevertheless, several interesting trends can be observed.

In terms of *energy consumption*, the SimpleConsolidator delivers one of the best results for each trace. For almost all traces, the results of ABC and GA are also among the best—the only exception is the nordugrid trace, for which the SimpleConsolidator delivers significantly better results than any other algorithm. The results of the PSO and FFBFD algorithms rarely belong to the best; on the other hand, they are also rarely more than 10% worse than the best results.

Concerning the *duration of the simulation*, the two single-solution heuristics (SC and FFBFD) are usually best. As expected, the population-based algorithms (ABC, GA, PSO) are usually slower, sometimes much slower (for instance, in the case of the sharcnet trace), since these consolidators have much higher execution time. The population-based algorithms exhibit quite large variance in their running time, in the sense that different population-based consolidators or even the same consolidator but with different PM schedulers can lead to significantly different simulation times on the same workload trace. This is due to the fact that we tuned each pair of consolidator and PM scheduler individually for a trace, and as noted in Section 5.3, this often led to the adoption of very different parameter configurations. Choosing different values for the population size or the number of iterations directly translates to different algorithm running times.

Regarding the *number of migrations*, the FFBFD algorithm usually leads to significantly fewer migrations than the other algorithms. Among the other algorithms, there is no clear tendency of which algorithm would lead to fewer or more migrations.

It could be assumed that, to achieve lower energy consumption, a more "thorough" optimization is necessary, requiring longer simulation time (because of the longer algorithm execution time) and more migrations. However, the results exhibit no such correlation. Low energy consumption can often be achieved with low simulation time and few migrations. In addition, high simulation time and a large number of migrations often lead to bad energy consumption values.

Concerning the effect of the used *PM scheduler*, no systematic difference can be determined from the results. That is, in some cases, using the SDM scheduler leads to better results, while in other cases the CF scheduler proved better.

In the overall comparison of the algorithms, it becomes clear that the SimpleConsolidator offers in most cases a very good trade-off between the considered metrics. The GA and ABC algorithms lead in most cases to similarly good energy consumption values as the SimpleConsolidator. Looking at only the energy consumption and the number of migrations, there is no clear winner from these three algorithms. However, in terms of simulation time, the SimpleConsolidator performs usually clearly better than GA and ABC. This was expected, but the insight from the results is that GA and ABC do not offer a significant gain in terms of solution quality (energy consumption or number of migrations) to compensate for the increased simulation time. The performance of the PSO algorithm was rather disappointing: in most cases, the PSO is outperformed by several other algorithms concerning all considered metrics. We suspect this is because—at least with the used solution encoding—the arithmetics performed by PSO to change the particle locations is not very meaningful, only very rarely leading to improvements. Finally, the FFBFD algorithm leads to few migrations and is also quite fast, but its results are not that good in terms of energy consumption. Hence, FFBFD can be recommended mainly in environments where migration is particularly costly.

6 | CONCLUSIONS

In this article, we addressed the problem of evaluating and comparing algorithms for data center consolidation. Though several algorithms had been proposed for consolidating VMs in data centers, only a few were comprehensively evaluated.

We presented an extension of the DISSECT-CF simulator to foster the implementation and evaluation of data center consolidation algorithms. On this basis, we implemented five consolidation algorithms: two single-solution heuristics and three population-based metaheuristics, which use the same solution encoding, the same fitness function, and the same search improvement techniques. We compared first different parameter configurations of each algorithm, and then the best found configuration of each algorithm, on 14 large-scale real-world workload traces. Based on the results of the comparison, we can draw the following conclusions:

- For different workload traces, different parameter configurations should be used to obtain good results.
- Local search is beneficial to boost the performance of the population-based metaheuristics.
- There is no clear winner among the algorithms that would consistently deliver good results regarding all considered quality metrics and for all traces.
- In terms of energy consumption, the SimpleConsolidator, the GA, and the ABC algorithm delivered the best results for most traces.
- The population-based metaheuristics exhibit significantly higher execution time than the single-solution heuristics, leading to also considerably higher simulation time. In particular, GA and ABC are much slower than the SimpleConsolidator and do not offer a significant gain in terms of solution quality (energy consumption or number of migrations) to compensate for the increased simulation time.

- The FFBFD algorithm leads consistently to the fewest migrations and is also fast; however, its results in terms of energy consumption are usually outperformed by the other algorithms. Hence, FFBFD can be recommended mainly in environments where migration is particularly costly.
- The PSO algorithm is usually outperformed by several other algorithms.

In more general terms, our results suggest that, at least for the given variant of the VMCP, population-based metaheuristics do not offer a clear benefit over a carefully engineered custom single-solution heuristic. Moreover, the GA and ABC algorithms which work with the logical structure of candidate solutions seem to be more appropriate for the VMCP, at least with the considered solution encoding, than the PSO algorithm which uses linear multidimensional arithmetics on the candidate solutions.

The evaluation of the impact of different solution encodings on the effectiveness of different VM consolidation algorithms is a possible path for future research. Another important topic for future research is the parallelization of population-based VM consolidation algorithms and their experimental evaluation.

ACKNOWLEDGMENT

The work of R. Ponto and Z. Á. Mann was partially supported by the European Union's Horizon 2020 research and innovation program under grant 731678 (RestAssured). Open access funding enabled and organized by Projekt DEAL.

ORCID

Zoltán Á. Mann https://orcid.org/0000-0001-5741-2709

REFERENCES

1. Mell P, Grance T. The NIST definition of cloud computing. National Institute of Standards and Technology, SP 800-145; 2011.
2. 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. *Concurr Comput Pract Exper*. 2012;24(13):1397-1420.
3. Beloglazov A, Abawajy J, Buyya R. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Futur Gener Comput Syst*. 2012;28(5):755-768.
4. Mann ZÁ. Modeling the virtual machine allocation problem. Proceedings of the International Conference on Mathematical Methods, Mathematical Models and Simulation in Science and Engineering, Vienna, Austria; 2015:102-106.
5. Mann ZÁ. Approximability of virtual machine allocation: much harder than bin packing. Proceedings of the 9th Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications, Fukuoka, Japan; 2015:21-30.
6. Mann ZÁ, Szabó M. Which is the best algorithm for virtual machine placement optimization? *Concurr Comput Pract Exper*. 2017;29(10):e4083.
7. Bobroff N, Kochut A, Beaty K. Dynamic placement of virtual machines for managing SLA violations. Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Network Management, Munich, Germany; 2007:119-128.
8. Wood T, Shenoy P, Venkataramani A, Yousif M. Sandpiper: black-box and gray-box resource management for virtual machines. *Comput Netw*. 2009;53(17):2923-2938.
9. Hallawi H, Mehnen J, He H. Multi-capacity combinatorial ordering ga in application to cloud resources allocation and efficient virtual machines consolidation. *Futur Gener Comput Syst*. 2017;69:1-10.
10. Jiang J, Feng Y, Zhao J, Li K. DataABC: a fast abc based energy-efficient live VM consolidation policy with data-intensive energy evaluation model. *Futur Gener Comput Syst*. 2017;74:132-141.
11. 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*. 2016;98(3):303-317.
12. Shi L, Furlong J, Wang R. Empirical evaluation of vector bin packing algorithms for energy efficient data centers. Proceedings of the IEEE Symposium on Computers and Communications, Split, Croatia; 2013:9-15.
13. Kecskemeti G. DISSECT-CF: a simulator to foster energy-aware scheduling in infrastructure clouds. *Simul Model Pract Theory*. 2015;58:188-218.
14. Mann ZÁ. Cloud simulators in the implementation and evaluation of virtual machine placement algorithms. *Softw Pract Exper*. 2018;48(7):1368-1389.
15. Ahmad RW, Gani A, Hamid SHA, Shiraz M, Yousafzai A, Xia F. A survey on virtual machine migration and server consolidation frameworks for cloud data centers. *J Netw Comput Appl*. 2015;52:11-25.
16. Silva Filho MC, Monteiro CC, Inácio PR, Freire MM. Approaches for optimizing virtual machine placement and migration in cloud environments: a survey. *J Parall Distrib Comput*. 2018;111:222-250.
17. Calheiros RN, Ranjan R, Beloglazov A, De Rose CA, Buyya R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw Pract Exper*. 2011;41(1):23-50.
18. Abdullah M, Lu K, Wieder P, Yahyapour R. A heuristic-based approach for dynamic VMs consolidation in cloud data centers. *Arab J Sci Eng*. 2017;42:3535-3549.
19. Kertész A, Dombi JD, Benyi A. A pliant-based virtual machine scheduling solution to improve the energy efficiency of IaaS clouds. *J Grid Comput*. 2016;14(1):41-53.
20. Kecskemeti G. DISSECT-CF website; 2019. https://github.com/kecskemeti/dissect-cf/. Online accessed March, 2019.
21. Moghaddam FF, Moghaddam RF, Cheriet M. Carbon-aware distributed cloud: multi-level grouping genetic algorithm. *Clust Comput*. 2015;18(1):477-491.
22. Zheng Q, Li R, Li X, Wu J. A multi-objective biogeography-based optimization for virtual machine placement. Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Shenzhen, China; 2015:687-696.
23. Feller E, Morin C, Esnault A. A case for fully decentralized dynamic VM consolidation in clouds. Proceedings of the IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom), Taipei, Taiwan; 2012:26-33.
24. Mitchell M. *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press; 1998:.

25. Karaboga D. An Idea Based on Honey Bee Swarm for Numerical Optimization. Technical Report, TR06, Erciyes University, Engineering Faculty, Computer Engineering Department; 2005.
26. Kennedy R, Eberhart R. Particle swarm optimization. Proceedings of IEEE International Conference on Neural Networks (ICNN), Perth, Australia; 1995.
27. Li Z, Yan C, Yu L, Yu X. Energy-aware and multi-resource overload probability constraint-based virtual machine dynamic consolidation method. *Futur Gener Comput Syst*. 2018;80:139-156.
28. Liu XF, Z.-H. Zhan, K.-J. Du, and W.-N. Chen. Energy aware virtual machine placement scheduling in cloud computing based on ant colony optimization approach. In Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation (GECCO), pages 41–48, 2014.
29. Farahnakian F, Ashraf A, Liljeberg P, et al. Energy-aware dynamic VM consolidation in cloud data centers using ant colony system. *In IEEE 7th International Conference on Cloud Computing (CLOUD), pages*. 2014;104-111.
30. Duan H, Chen C, Min G, Wu Y. Energy-aware scheduling of virtual machines in heterogeneous cloud computing systems. *Futur Gener Comput Syst*. 2017;74:142-150.
31. Farahnakian F, Ashraf A, Pahikkala T, et al. Using ant colony system to consolidate VMs for green cloud computing. *IEEE Trans Serv Comput*. 2015;8(2):187-198.
32. Aryania A, Aghdasi HS, Khanli LM. Energy-aware virtual machine consolidation algorithm based on ant colony system. *J Grid Comput*. 2018;16:477-491.
33. Zheng Q, Li R, Li X, et al. Virtual machine consolidated placement based on multi-objective biogeography-based optimization. *Futur Gener Comput Syst*. 2016;54:95-122.
34. Kansal NJ, Chana I. Energy-aware virtual machine migration for cloud computing-a firefly optimization approach. *J Grid Comput*. 2016;14(2):327-345.
35. Deng W, Liu F, Jin H, Liao X, Liu H, Chen L. Lifetime or energy: consolidating servers with reliability control in virtualized cloud datacenters. IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom); 2012:18-25.
36. Qiu W, Qian Z, Lu S. Multi-objective virtual machine consolidation. Proceedings of the IEEE 10th International Conference on Cloud Computing (CLOUD), Honolulu, Hawaii; 2017:270-277.
37. Shi T, Ma H, Chen G. Energy-aware container consolidation based on pso in cloud data centers. Proceedings of the 2018 IEEE Congress on Evolutionary Computation (CEC), Rio de Janeiro, Brazil; 2018:1-8.
38. Armant V, De Cauwer M, Brown KN, O'Sullivan B. Semi-online task assignment policies for workload consolidation in cloud computing systems. *Futur Gener Comput Syst*. 2018;82:89-103.
39. V. De Maio, G. Kecskemeti, and R. Prodan. An improved model for live migration in data centre simulators. Proceedings of the 9th International Conference on Utility and Cloud Computing (UCC), Shanghai, China; 2016:108-117.
40. Kecskemeti G, Markus A, Kertesz A. Cost-efficient datacentre consolidation for cloud federations. Proceedings of the 8th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER; 2018:213-220; INSTICC, SciTePress. https://doi.org/10.5220/0006775302130220.
41. Dengiz B, Altiparmak F, Smith AE. Local search genetic algorithm for optimal design of reliable networks. *IEEE Trans Evol Comput*. 1997;1(3):179-188.
42. Mernik M, Liu S-H, Karaboga D, Črepinšek M. On clarifying misconceptions when comparing variants of the artificial bee colony algorithm by offering a new implementation. *Inf Sci*. 2015;291:115-127.
43. Lemieux C. *Monte Carlo and Quasi-Monte Carlo Sampling*. New York, NY: Springer; 2009.

How to cite this article: Ponto R, Kecskeméti G, Mann ZÁ. Comparison of workload consolidation algorithms for cloud data centers. *Concurrency Computat Pract Exper*. 2021;33:e6138. https://doi.org/10.1002/cpe.6138