Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

BalCon — resource balancing algorithm for VM consolidation

Andrei Gudkov[1], Pavel Popov[1], Stepan Romanov[*,1]

Huawei Technologies Company Ltd, Russian Research Institute, Moscow, 121099, Russian Federation

A R T I C L E   I N F O

Article history:
Received 28 September 2022
Received in revised form 18 April 2023
Accepted 5 May 2023
Available online 18 May 2023

Dataset link: https://github.com/andreigudkov/BalCon

Keywords:
Consolidation
Bin packing
High-multiplicity
Cloud computing
Resource scheduling
Heuristic algorithms

A B S T R A C T

Cloud providers handle substantial number of requests to create and delete virtual machines (VMs) on a daily basis, where the unknown sequence of requests eventually leads to resource fragmentation. To mitigate this issue, periodic consolidation of VMs into fewer number of physical hosts is an important cost-saving procedure, closely related to the vector bin-packing problem. In this paper, we propose the BalCon algorithm for consolidation that steadily reduces the number of active hosts and keeps migration costs low. BalCon classifies the cluster's state and selects one of three heuristics to balance resources for superior consolidation. To evaluate BalCon's performance with respect to optimality, we introduce integer programming models. BalCon finds 99.7% of the optimal solutions for over 750 problem instances. This outstanding result was achieved due to the Force Step of our algorithm, which is the key improvement detail for common heuristics. We compare BalCon with a modified Sercon heuristic using Huawei and synthetic datasets with two resources for allocation.

1. Introduction

Virtualization technology allows the sharing of a physical machine (host) between multiple isolated virtual machines (VMs) [1,2]. Resources for virtualization include the central processing unit (CPU), random access memory (RAM), network, drives, *etc.* Several users can run VMs with different operating system (OS) on the same host simultaneously. The resource sharing prospect of virtualization opens up variety of opportunities for business.

Cloud providers operate datacenters with thousands of hosts and exploit virtualization to lease VMs to clients [3,4]. Resources combinations for the VMs are usually predefined and known as flavors. For instance, two dimensional flavors (CPU, RAM) can be (2 cores, 4 GiB), (8 cores, 24 GiB), (64 cores, 560 GiB), *etc.* Clients choose a flavor and an OS image for the VM and remotely use the VM on a "pay-as-you-go" basis. The guarantees from providers to clients regarding VMs availability and reliability are specified in a service level agreement (SLA) [5,6], where violation leads to the providers being penalized. Therefore, during datacenters operation the providers strive to minimize SLA violations and operational costs.

Hosts in an idle state consume about 70% of maximal power consumption [7]. To minimize operational costs, providers seek to allocate VMs into the fewest number of hosts and power off the unused ones. The scope for optimization arises from the uncertainty of requests during cloud operation, which leads to fragmentation of resources.

The VMs in datacenters are managed through three essential operations: (1) create VM on host, (2) delete VM from host, and (3) migrate VM from one host to another. The periodic deletion of VMs (see Fig. 1a) along with unpredictable requests to create VMs (see Fig. 1b) leads to resource fragmentation on the hosts and non-optimal VM placement.

Live migration is an important technique for the optimization of datacenters, as it allows running VMs to be migrated between hosts without interruption or client notice [8]. However, frequent live migration is undesirable due to the increased load on the management side of the cloud, where the load depends on various factors, such as the state of the VMs (processes, memory, network, *etc.*). Therefore, it is preferable to migrate less active VMs with less RAM.

1.1. Consolidation

Consolidation of virtual machines [9] is a procedure that searches for a better assignment of VMs to hosts. The improved assignment benefits one or more of the following metrics: power consumption [7,10], number of active hosts [11,12], migration cost [11,13–15], resource fragmentation [12,16], number of SLA violations [11], *etc.* Typically, minimization of the number of active hosts is a primary goal of consolidation, however taking migration cost into account is also crucial. Indeed, large number

* Corresponding author.
E-mail addresses: gudkov.andrei@huawei.com (A. Gudkov), popov.pavel@huawei.com (P. Popov), stepan.romanov@huawei.com (S. Romanov).

1 The authors equally contributed to this work.

https://doi.org/10.1016/j.future.2023.05.001
0167-739X/© 2023 Elsevier B.V. All rights reserved.

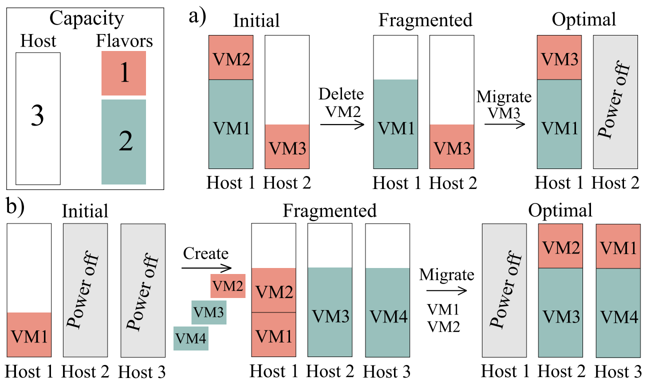

Fig. 1. An example of resource fragmentation caused by delete and create operations followed by optimization via the migration operation. Panel (a) demonstrates how a delete operation of VM2 led to resource fragmentation, which was optimized by migration of VM3. Panel (b) shows how a sequence of create operation for the trace of VMs (VM2, VM3, VM4) by the First Fit heuristic caused resource fragmentation, which was optimized by the migration of VM1 and VM2.

of migrations introduce undesirable loads on the management side of a datacenter. To avoid overloads, a balance between migration cost and the number of active hosts must be preserved.

Consolidation relies on two major approaches: static and dynamic [9]. Static consolidation reserves resources for a VM based on its maximal consumption, which is specified in the flavor. Dynamic consolidation assigns VMs to hosts using either their current or predicted, by statistical model, resource requirements of VMs. These requirements are usually lower than the flavor size. In other words, static consolidation guarantees the availability of the resources on the host for the clients, whereas dynamic consolidation provides denser packing at the cost of possible SLA violations — if demand of VMs on a host rises to a higher level.

Furthermore, algorithms for consolidation are divided into centralized [9,10] and decentralized [17–19] depending on the cloud architecture. In centralized algorithms, any host possesses full information about the entire cluster. In decentralized algorithms, all hosts pursue the same goal, however each host has information only about its neighborhood. Consolidation for centralized architectures works faster and provides better results, whereas decentralized architectures are free from a single point of failure and have better scalability.

1.2. d-dimensional vector bin packing

Consolidation of VMs is closely related to the well-known *d*-dimensional vector bin packing (d-VBP) problem. In the d-VBP problem, items are represented as d-dimensional vectors which are packed into the fewest number of d-dimensional bins such that: (1) all items are placed (2) the sum of items in each bin is less than or equal to the bin capacity in all dimensions. The d-VBP problem is known to be an NP-hard problem which is exponentially hard to solve [20]. However, if $d = 1$, there are two types of restrictions of the d-VBP problem that are often applicable to consolidation and lead to optimal solutions in polynomial time: (1) divisible items – in a list of sorted items each item is a divisor of next items – can be optimally packed with the First Fit Decreasing heuristic [21], (2) high-multiplicity VBP problem – items form a few groups and the items in each group are identical – can be solved in polynomial time [22]. To the best of our knowledge, the d-VBP problem for $d \neq 1$ is still exponentially complex even with the above restrictions.

The hardness of the d-VBP problem requires the use of heuristics to obtain a solution in reasonable time [23–25]. Unfortunately, the d-VBP misses objective of minimizing of migration cost, which restricts its application to migration-aware consolidation [14]. Application of d-VBP heuristics to the consolidation problem is equivalent to removing all the VMs from the hosts and placing them again. Such actions require unacceptably large amounts of memory for migration.

1.3. Methods for solving the consolidation problem and proposed approach

To solve the multidimensional consolidation problems researchers mainly utilize three methods: (1) integer linear programming (ILP), (2) metaheuristics, and (3) heuristics. ILP formulations obtain the optimal solution only for the small problem instances because of the exponential time complexity of solvers [26–28]. Metaheuristics work faster than ILP solvers and provide approximate solutions for larger instances. Such metaheuristics algorithms include: genetic algorithms [29,30], ant colony optimization [16,31], particle swarm optimization [32], artificial bee colony [33], *etc.* Typically, the performance of metaheuristics is worse than that of the heuristics in terms of the solvable instance size, parameters dependence, and time complexity [34].

Heuristics for the consolidation problem practically used due to time efficiency, small number of parameters, and implementation simplicity. The majority of migration-aware heuristics rely on this common approach: sort hosts, sort VMs, and try to place largest VM into the most loaded host [11,12,14,35]. If the VM does not fit into a host because of a lack of free space, then the heuristic skips the host and try the next one. We call such skipping heuristics that exploit only available free space — "Sercon-like". The heuristics differ in the criteria for sorting of d-dimensional objects, additional classifications, and algorithm parameters. *Murtazaev* and *Oh* performed sorting using "surrogate weight" and introduced parameters for the maximum number of migrations and minimum migration efficiency [14]. *Ferreto et al.* used lexicographic order for sorting and held steady VMs [11]. *Rao* and *Thilagam* proposed hosts classification into receivers and donors based on the theoretical minimum of required hosts and introduced a defragmentation procedure to reduce resource fragmentation after consolidation [12].

The main disadvantage of Sercon-like heuristics becomes evident in consolidation problems with 2 resources. The heuristics are poorly adapted to solving instances with imbalanced hosts lacking free space for one of the resources (see Fig. 2a, RAM limits Host 2, whereas CPU limits Host 3). Sercon-like heuristics are unable to free any of the hosts from the assignment demonstrated in Fig. 2a. However, migrations of yellow and green VMs between Host 2 and Host 3 increase free space and release Host 1 (see Fig. 2b).

In this paper, we propose the BalCon algorithm for solving 2D centralized consolidation problems with a controlled tradeoff between the number of released hosts and the amount of migrated memory. The key component of BalCon is the Force Step, which places a VM into the host with a lack of free space using induced migrations. For instance, Fig. 2c illustrates two Force Steps that are required to consolidate the initial assignment (Fig. 2a) into the optimal assignment (Fig. 2b). To efficiently place 2D VMs, we introduce the balance factor to classify the cluster state and choose one of three heuristic for VM placement. To evaluate BalCon we use two datasets: real VM assignments from Huawei Cloud datacenters and synthetically generated hard instances for the purpose of stress testing. Also we introduce ILP models and a modified the Sercon heuristic for comparison. The results indicate the superior performance of the BalCon towards solving consolidation problems under resource imbalance. The code with the implementation of BalCon and the synthetic datasets are presented in https://github.com/andreigudkov/BalCon.

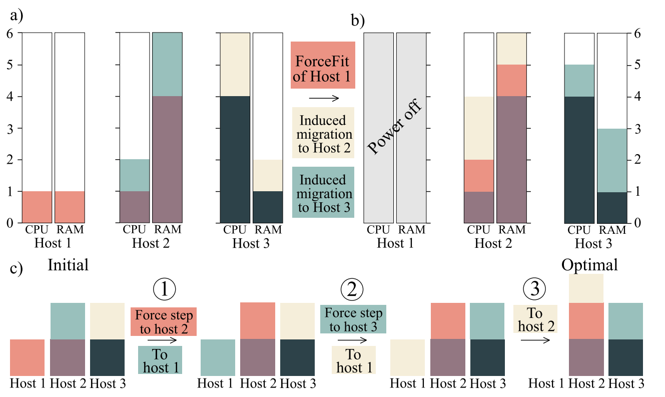

Fig. 2. A consolidation problem instance with two resources (CPU, RAM) which is solvable by BalCon and unsolvable by Sercon-like heuristics. The problem involves four color-coded VMs that need to be assigned to three hosts, each with a capacity of (6, 6) *a.u.* Panel (a) shows the initial assignment of VMs to hosts. Panel (b) shows the optimal assignment of VMs to hosts. Panel (c) shows the sequence of actions taken by the BalCon algorithm to optimize the initial assignment panel (a) to the final assignment panel (b).

This paper is organized as follows. In Section 2, we formulate the cloud consolidation problem. In Section 3, we introduce integer programming models that were used to obtain optimal solutions and lower bounds for performance evaluation. In Section 4, we provide a description of the BalCon algorithm, including the basic ideas (as shown in the diagram in Section 4.1) and the formal pseudo code (as presented in Section 4.3). In Section 5, we describe the datasets and compare BalCon with Sercon and optimal solutions. In Section 6, we conclude the paper and propose possible directions for future work.

2. Problem statement

In our scenario, we operate a cluster with sets of physical hosts $h \in H$, VMs $v \in V$, and flavors $f \in F$. Every host h provides a pair of integer resources for scheduling: CPU (cores) and RAM (tebibytes). We denote the capacity of host h as $(h.cpu, h.mem)$, and the capacity of other resources is denoted in the same way. The demand for resources of a VM v is defined in a flavor f, that is, $(v.cpu, v.mem) = (f.cpu, f.mem)$. Each host h reserves resources for the allocated VM v.

Definition 1. A mapping μ is the assignment of VMs to hosts, where some VMs can be unassigned. The mapping μ is feasible if:

1. every VM is assigned to exactly one host.
2. capacity of all hosts is satisfied in both (CPU and RAM) dimensions:

$$\forall h \in H,\ \sum_{v \in h} v.cpu \le h.cpu \ \wedge\ \sum_{v \in h} v.mem \le h.mem$$

Our goal is to consolidate the VMs of the cluster. Given an initial feasible mapping μ_0 of the VMs to the hosts, we try to compute a different feasible mapping μ_b that minimizes the number of active hosts and the amount of migrated memory.

Definition 2. For a given mapping μ, host h is called *active* if it contains at least one VM v. $\exists v : \mu(v) = h$

Definition 3. For a given mapping μ, a VM v is called *migrated* if it changes host: $\mu(v) \ne \mu_0(v)$

To control our preference between the number of active hosts $A(\mu_b)$ and the amount of migrated RAM $M(\mu_b, \mu_0)$, we minimize the following linear objective function:

$$Obj(\mu_b, \mu_0) = w_a \cdot A(\mu_b) + w_m \cdot M(\mu_b, \mu_0) \tag{1}$$

where w_a, w_m are non-negative weights. It is important to note that the resulting mapping of the minimization problem depends on the ratio of the weights instead of their absolute values. To investigate the algorithms performance under different objective functions, we introduce the maximal memory for migration per host

$$MPH = \frac{w_a}{w_m} \text{ TiB.} \tag{2}$$

The parameter is used for evaluation (as described in Section 5.2). Additionally, in our algorithm, if the migration cost to release any host exceeds *MPH*, then releasing the host decreases the objective function. In the case of the classical VBP problem, $MPH = \infty$ TiB.

3. Integer programming solutions

3.1. Allocation model

Bartók and Mann [26] presented a straightforward Integer Linear Programming (ILP) formulation of the similar consolidation problem. However, our problem statement differs in the calculation of migration cost. Their formulation counts the number of migrated VMs, whereas we compute the total amount of migrated memory. The updated ILP allocation model is expressed in our terms as follows.

The binary variables are:

$$\begin{aligned}\mathrm{Alloc}_{v,h} &= 1 \text{ if } v \text{ is allocated on } h \text{ and 0 otherwise}\\ \mathrm{Active}_h &= 1 \text{ if } h \text{ is active and 0 otherwise}\\ \mathrm{Migr}_v &= 1 \text{ if } v \text{ is migrated and 0 otherwise}\end{aligned}$$

With these variables the objective is to

$$\text{minimize} \quad w_a \sum_{h \in H} \mathrm{Active}_h + w_m \sum_{v \in V} v.mem \cdot \mathrm{Migr}_v. \tag{3}$$

In other words, our objective is to minimize the number of active hosts and the amount of migrated memory, subject to the following constraints:

$$\sum_{h \in H} \mathrm{Alloc}_{v,h} = 1 \qquad \forall v \in V \tag{4}$$

$$\sum_{v \in V} v.cpu \cdot \mathrm{Alloc}_{v,h} \le h.cpu \qquad \forall h \in H \tag{5}$$

$$\sum_{v \in V} v.mem \cdot \mathrm{Alloc}_{v,h} \le h.mem \qquad \forall h \in H \tag{6}$$

$$\mathrm{Alloc}_{v,h} \le \mathrm{Active}_h \qquad \forall (v,h) \in V \times H \tag{7}$$

$$\mathrm{Migr}_v = 1 - \mathrm{Alloc}_{v,\mu_0(v)} \qquad \forall v \in V \tag{8}$$

Constraint (4) requires that every VM is scheduled on exactly one host (Definition 1.1). Resource Constraints (5) and (6) ensure that the CPU and RAM capacities of the hosts are satisfied (Definition 1.2). Constraint (7) guarantees that hosts with VMs are active (Definition 2). According to Constraint (8), a VM is migrated if it changes its host (Definition 3).

3.2. Flavor flow model

We introduce the flavor flow model that takes advantage of the fewer number of flavors than VMs. The allocation model uses $|H|$ variables for each VM, however many of the VMs are of the same flavor and have equal resource demands $(f.cpu, f.mem)$.

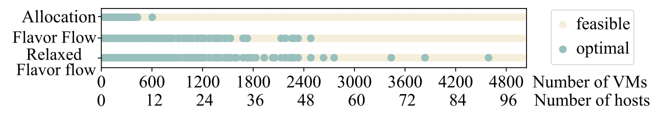

Fig. 3. Comparison of the ILP models on progressively larger instance sizes. The problem instances of different sizes were solved by the CBC solver for 10 min each. The mean number of flavors per problem instance was 27.

Permutation of these similar VMs leads to the same values of objective function and an undesirable symmetry. Instead of defining the exact location for every VM, we keep track on the flow of the VMs of the same flavor between hosts. Such a flavor flow model breaks the symmetry of the allocation model because all VMs of the same flavor are equal.

Formal definition of the flavor flow model requires defining the following variables:

$\mathrm{In}_{f,h} \in \mathbb{Z}$ is the number of VMs of flavor f migrated into host h

$\mathrm{Out}_{f,h} \in \mathbb{Z}$ is the number of VMs of flavor f migrated out from host h

$\mathrm{Active}_h = 1$ if h is active and 0 otherwise

With these variables the objective is to

$$\text{minimize} \quad w_a \sum_{h\in H} \mathrm{Active}_h + w_m \sum_{f\in F, h\in H} f.mem \cdot \mathrm{Out}_{f,h}. \tag{9}$$

Let $n_{f,h}$ be the number of VMs of flavor f initially located on host h, according to the mapping μ_0. Then the constraints of the flavor flow model are:

$$\sum_{h\in H} \mathrm{Out}_{f,h} = \sum_{h\in H} \mathrm{In}_{f,h} \qquad \forall f \in F \tag{10}$$

$$\mathrm{Out}_{f,h} \le n_{f,h} \qquad \forall (f,h) \in F \times H \tag{11}$$

$$n_{f,h} \cdot (1 - \mathrm{Active}_h) \le \mathrm{Out}_{f,h} \qquad \forall (f,h) \in F \times H \tag{12}$$

$$\sum_{f\in F} f.cpu \cdot (n_{f,h} + \mathrm{In}_{f,h} - \mathrm{Out}_{f,h}) \leqslant h.cpu \cdot \mathrm{Active}_h \ \forall h \in H \tag{13}$$

$$\sum_{f\in F} f.mem \cdot (n_{f,h} + \mathrm{In}_{f,h} - \mathrm{Out}_{f,h}) \leqslant h.mem \cdot \mathrm{Active}_h \ \forall h \in H \tag{14}$$

Constraint (10) ensures that the net flow of VMs is zero for each flavor f. In other words, the number of VMs migrated out from all hosts must be equal to the number of VMs migrated onto all hosts for each flavor f. Constraint (11) forbids moving out of a host more VMs of a flavor f than were initially present on the host. For inactive hosts, all VMs must be moved out according to Constraint (12). Resource Constraints (13) and (14) ensure that the CPU and RAM capacities of each host are satisfied.

The flavor flow model has asymptotically $|V|/|F|$ fewer variables than the allocation model, which allows for solving much larger problem instances when $|F| \ll |V|$ — a typical real cloud situation.

3.3. Lower bounds

We detected that the running time of ILP solvers is sensitive to the value of *MPH*. When *MPH* is close to 0 or infinity, solvers manage to find the optimal solutions μ_{opt} in a reasonable time. Unfortunately, for intermediate values of *MPH*, solvers are unable to provide the optimal solutions even in a few days. Nevertheless, relaxation of the flavor flow model allows one to obtain a lower bound (LB) μ_{LB} on the optimal solution for the intermediate *MPH* values. To perform the relaxation, we kept the $|H|$ variables Active_h as binary and relaxed the domain of the $2|F||H|$ variables $In_{f,h}$ and $Out_{f,h}$ to $\mathbb{R}$. This relaxation significantly simplifies the flow model for the solver and sets the lower bound on the optimal solution

$$Obj(\mu_{LB}, \mu_0) \le Obj(\mu_{opt}, \mu_0). \tag{15}$$

Table 1
Comparison of the models by domain of variables and number of constraints.

Model	Domain of variables	Number of constraints
Allocation	$\mathbb{B}^{\lvert V\rvert\lvert H\rvert+\lvert V\rvert+\lvert H\rvert}$	$\lvert V\rvert\lvert H\rvert + 2\lvert H\rvert + 2\lvert V\rvert$
Flavor flow	$\mathbb{B}^{\lvert H\rvert} \times \mathbb{Z}^{2\lvert F\rvert\lvert H\rvert}$	$2\lvert F\rvert\lvert H\rvert + 2\lvert H\rvert + \lvert F\rvert$
Relaxed Flavor flow (Lower Bound)	$\mathbb{B}^{\lvert H\rvert} \times \mathbb{R}^{2\lvert F\rvert\lvert H\rvert}$	$2\lvert F\rvert\lvert H\rvert + 2\lvert H\rvert + \lvert F\rvert$

The comparison of the allocation, flavor flow, and relaxed flavor flow models in terms of variable domain and number of constraints is presented in Table 1. To compare the performance of the models, we searched for the optimal solution of problem instances with different sizes using the CBC solver [36] for 10 min (see Fig. 3). As expected, the flavor flow models provided the optimal solutions for much larger instances than the allocation model.

4. Algorithms

4.1. Basic description of BalCon

The BalCon algorithm was created to cope with imbalanced situations in a datacenter. The need for such an algorithms arises from the poor performance of Sercon-like heuristics in terms of host balancing. For example, Sercon-like heuristics are unable to consolidate the imbalanced mapping from Fig. 2a because of insufficient amount of free space in the host for one of the resources (RAM limits Host 2, whereas CPU limits Host 3). However, the consolidation is possible by shuffling the yellow and green VMs between Host 2 and Host 3 (see Fig. 2b). The shuffling in the BalCon is implemented with Force Steps, which free space in a host by ejecting VMs from the host into a temporary buffer (stash S). The VMs from the stash are later moved into other hosts, which leads to induced migrations. To perform the consolidation in the example, BalCon uses two Force Steps. The sequence of actions for the consolidation is presented in Fig. 2c, where Host 1 illustrates the role of the stash. In the first step, to place the red VM to Host 2 we eject the green VM from Host 2. In the second step, to place the green VM to Host 3 we eject the yellow VM from Host 3. Finally, we place the yellow VM into the free space of Host 2. As a result, we released Host 1 and induced two migrations: (1) the yellow VM was migrated from Host 3 to Host 2 and (2) the green VM was migrated from Host 2 to Host 3.

In general, BalCon is a greedy heuristic that attempts to release one active host in each step. The basic ideas of the algorithm are presented in Fig. 4a, whereas a formal definition will be given in Section 4.3. In each iteration, BalCon chooses the smallest host according to migration cost and places a VM from the host into the stash S.

Definition 4. The migration cost of VM v is $v.mem$, and the migration cost of host h is $\sum_{w\in h} w.mem$, where w are unmigrated VMs: $\mu_0(w) = h$.

Definition 5. The stash S is a temporary buffer for VMs v with a resource vector $s = \left(\sum_{v\in S} v.cpu, \sum_{v\in S} v.mem\right)$

For the success of a host release, S must be emptied. The BalCon takes the largest VM from the stash and uses the Best Fit

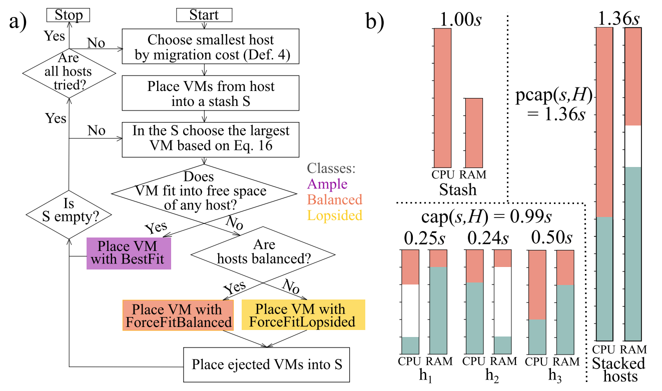

Fig. 4. (a) A diagram illustrates the workflow of the BalCon algorithm (b) An illustration of the calculation of cap (Eq. (20)) and pcap (Eq. (21)) for the balance factor. The stash resource vector s (red color) represents a unit vector to measure the amount of the free space. The green color corresponds to the occupied space in the hosts by VMs.

heuristic until it encounters a problem with free space on all the hosts. The largest VM v is defined using the following formula:

$$|v| = \frac{v.cpu}{\sum_{w \in V} w.cpu} + \frac{v.mem}{\sum_{w \in V} w.mem}. \tag{16}$$

The free space of a host h is defined by the mean of its load and allocated VMs $v \in h$

$$\mathrm{load}(h) = \left(\sum_{v \in h} v.cpu, \sum_{v \in h} v.mem \right) \tag{17}$$

$$\mathrm{free}(h) = (h.cpu - \mathrm{load}(h).cpu,\ h.mem - \mathrm{load}(h).mem) \tag{18}$$

In the case of a lack of free space on the hosts, BalCon classifies the situation using the balance factor (defined formally in Section 4.2) and chooses one of two heuristics with Force Steps: *ForceFitBalanced* or *ForceFitLopsided*. These heuristics pick a destination host h and free space for a given VM v by ejecting VMs from h into the stash. This procedure repeats and VMs from the stash are assigned to new or their original hosts. The emptying of the stash indicates the generation of a new feasible mapping. In the new mapping some of the ejected VMs could migrate between active hosts.

Definition 6. A migration of a VM is called *induced* if it takes place between active hosts.

Using induced migrations BalCon solves the problem of deficiency of free space in a datacenter. Induced migrations allow one to shuffle VMs and release the host, at the cost of an increase in the amount of migrated memory.

4.2. Balance factor

We introduce the balance factor as a measure of the distribution of free space in a datacenter. The factor allows us to estimate the potential for VM shuffling operation to increase VM placements. In each step, BalCon attempts to fit the stash S into the hosts. Therefore, the distribution of free space between the hosts is improved if more stashes can be accommodated by the hosts. The balance factor is defined as the ratio of the current amount of stashes in the free space of the hosts $\mathrm{cap}(s, H)$ to the potential amount of stashes in the combined free space of all hosts $\mathrm{pcap}(s, H)$

$$BF(s, H) = \frac{\mathrm{cap}(s, H)}{\mathrm{pcap}(s, H)}, \tag{19}$$

where

$$\mathrm{cap}(s, H) = \sum_{h \in H} \min \left(\frac{\mathrm{free}(h).cpu}{s.cpu}, \frac{\mathrm{free}(h).mem}{s.mem} \right) \tag{20}$$

$$\mathrm{pcap}(s, H) = \min \left(\sum_{h \in H} \frac{\mathrm{free}(h).cpu}{s.cpu}, \sum_{h \in H} \frac{\mathrm{free}(h).mem}{s.mem} \right). \tag{21}$$

The domain of $BF(s, H)$ is $[0, 1]$, where the value of 1 corresponds to a purely balanced situation without potential for improvement, whereas a value of 0 corresponds to a purely lopsided situation that can be improved. In the balanced case most hosts reach their limit by the same resource, unlike in the lopsided case where some hosts are limited by *cpu* and some by *mem*.

Note that both $\mathrm{pcap}(s, H)$ and $\mathrm{cap}(s, H)$ provide relaxed values that ignore the actual sizes of separate VMs in the stash. Thus, $\mathrm{pcap}(s, H)$ and $\mathrm{cap}(s, H)$ set upper bounds on the amount of stashes in any mapping and current mapping respectively. For instance, if $\mathrm{cap}(s, H) < 1$ the allocation of all VMs from the stash is impossible without induced migrations. Similarly, if $\mathrm{pcap}(s, H) < 1$ the stash is implacable into the hosts by any mapping. To distinguish between Balanced and Lopsided situations, we introduce the parameter α. If $BF(s, H) < \alpha$ or $\mathrm{cap}(s, H) < 1$ the situation is Lopsided and Balanced otherwise. In practice we use $\alpha = 0.95$.

An example of capacities calculation is presented in Fig. 4b. The stash with a resource capacity of $s = (8.0, 4.0)$ is used as a unit vector of free space. Additionally, there are three partially filled hosts with equal capacities of $(6.0, 6.0)$ and free spaces $\mathrm{free}(h_1) = (5.0, 1.0)$, $\mathrm{free}(h_2) = (1.9, 5.0)$, and $\mathrm{free}(h_3) = (4.0, 2.0)$. In total, the hosts can allocate at most

$$\begin{aligned}\mathrm{cap}(s, H) &= \min \left(\frac{5.0}{8.0}, \frac{1.0}{4.0} \right) + \min \left(\frac{1.9}{8.0}, \frac{5.0}{4.0} \right) + \\ &+ \min \left(\frac{4.0}{8.0}, \frac{2.0}{4.0} \right) = 0.25 + 0.24 + 0.50 = 0.99\end{aligned} \tag{22}$$

of the stashes without induced migrations. Potential capacity is computed by combining the three hosts together that results in

$$\begin{aligned}\mathrm{pcap}(s, H) &= \min \left(\frac{5.0 + 1.9 + 40}{8.0}, \frac{1.0 + 5.0 + 2.0}{4.0} \right) = \\ &= 1.36\end{aligned} \tag{23}$$

of the stashes sizes. Finally, we compute the balance factor $BF(s, H) = \mathrm{cap}(H)/\mathrm{pcap}(H) = 0.73$. The fact that the BF is lower than $\alpha = 0.95$ indicates a resource imbalance and potential for improvement by induced migrations.

4.3. The BalCon algorithm

The core structure of our algorithm is presented in the Listing 1. There are three global parameters in the algorithm:

- α is used as a threshold to classify imbalanced ($BF(s, H) < \alpha$) and balanced ($BF(s, H) \geq \alpha$) situations using Eq. (19).
- b is the maximal number of Force Steps to try during host release.
- γ is the maximal number of tries of the same destination host in a row.

The BalCon sorts hosts by migration cost (see Definition 4) and attempts to release the hosts one by one (rows 3–4). To release host h, the procedure searches for a better feasible mapping μ_{tmp} in rows 5–9. In the better mapping, h has to be turned off, and VMs from h are reassigned to active hosts A. The VMs from h are placed into the stash S and unassigned from the mapping μ_{tmp} in

Listing 1: High-level BalCon algorithm structure

```
Input              : $H$ is the list of hosts
                     $V$ is the list of VMs
                     $\mu_0$ is the initial feasible mapping
Global Parameters: $\alpha = 0.95$
                     $b = 4000$
                     $\gamma = 3$
Output             : At the end, $\mu_b$ is a feasible mapping
                     of VMs to hosts
1  Procedure BalCon($H$, $V$, $\mu_0$)
2      $\mu_b$ := copy $\mu_0$
3      $H$ := sort $H$ by migration cost
4      for $h$ in $H$ do
5          $\mu_{tmp}$ := copy $\mu_b$
6          $S$ := get VMs from $h$ according to $\mu_{tmp}$
7          $\mu_{tmp}$ := Unassign all VMs in $S$ from $\mu_{tmp}$
8          $A$ := get active hosts from $\mu_{tmp}$
9          $\mu_{tmp}$ := ForceFit($S$, $A$, $\mu_{tmp}$)
10         if $\mu_{tmp}$ is feasible and $Obj(\mu_{tmp}, \mu_0) \leq Obj(\mu_b, \mu_0)$
11             $\mu_b$ :=$\mu_{tmp}$
12     return $\mu_b$
```

Listing 2: ForceFit heuristic

```
1  Procedure ForceFit($S$, $A$, $\mu$)
2      $ForceSteps$ := 0
3      $p$ := RepeatsProhibitor($A$, $\gamma$)
4      while $S$ is not empty and $ForceSteps < b$ do
5          $v$ := peek largest $v$ in $S$
6          $class$ := Classify($S$, $A$, $\mu$, $v$)
7          $S$ := remove $v$ from $S$
8          if $class$ is "Ample"
9              $\mu$ := BestFit($v$, $A$, $\mu$)
10         if $class$ is "Balanced"
11             $ForceSteps$ +=1
12             $h, p$ := ChooseHostBalanced($v$, $A$, $\mu$, $p$)
13             $\mu, V_e$ := ForceFitBalanced($v$, $h$, $\mu$)
14             $S$ := add all VMs from $V_e$ to stash $S$
15         if $class$ is "Lopsided"
16             $ForceSteps$ +=1
17             $h, p$ := ChooseHostLopsided($v$, $A$, $\mu$, $p$)
18             $\mu, V_e$ := ForceFitLopsided($v$, $h$, $\mu$)
19             $S$ := add all VMs from $V_e$ to stash $S$
20     return $\mu$
21
22 Procedure Classify($S$, $A$, $\mu$, $v$)
23     if $\exists h \in A : v$ fits $h$
24         return "Ample"
25     $cap$ := Capacity($S$, $A$, $\mu$)
26     $pcap$ := PotentialCapacity($S$, $A$, $\mu$)
27     if $cap < 1$ or $cap < \alpha \cdot pcap$
28         return "Lopsided"
29     return "Balanced"
```

rows 6–7. In rows 9–11 ForceFit procedure tries to fit VMs from the stash S into active hosts A. If a new mapping μ_{tmp} is feasible and leads to an improvement of the objective function, then host h is successfully released (rows 10–11).

The ForceFit heuristic is presented in Listing 2. In rows 2–3, we define two variables for the procedure: *ForceSteps* and p. The integer variable *ForceSteps* is used to control the number of iterations in the while loop (rows 4–19). To avoid coming back to previously-visited solutions, we prohibit a host from being selected more than γ times in a row during Force Steps. The state information is maintained in the object instance p.

The ForceFit heuristic works until the stash is empty or the Force Steps limit is reached. In each step, ForceFit tries to insert the largest VM v (Eq. (16)) from the stash into the destination host h. The choice of the destination host depends on the cluster state, which is determined by the `Classify` procedure (rows 22–29). The procedure implements ideas from Section 4.2 and classifies the state into three classes: Ample (Section 4.4), Balanced (Section 4.5), and Lopsided (Section 4.6). The last two classes use Force Steps and differ in their approach to choose the destination host h and VMs V_e to eject from h.

The choice of VMs V_e influences the number of induced migrations and the amount of migrated memory at the end of ForceFit. To reduce the amount of migrated memory, we prefer to eject VMs V_e that were migrated to the destination host h during previous steps of the algorithm. The shuffling of these migrated VMs in ForceFit is unable to increase the objective function. In other words, only new migrations in ForceFit increase the objective function compared to the best step of BalCon μ_b. In terms of memory, improving the objective function (row 10 of Listing 1) requires the amount of new memory for migration during ForceFit to be no more than MPH ($M(\mu_{tmp}, \mu_0) - M(\mu_b, \mu_0) \leq MPH$).

4.4. Ample class

In the Ample class one or more hosts have enough free space to accommodate a VM. We employ a Best Fit heuristic to allocate VM v. The destination host h is chosen based on surrogate load

$$|h| = \frac{\text{load}(h).cpu}{h.cpu} + \frac{\text{load}(h).mem}{h.mem}. \tag{24}$$

Among all hosts with enough free space, we assign a VM to the host with the highest load.

4.5. Balanced class

To place a VM v in the Balanced class (Listing 2 rows 11–14), some other VMs must be ejected and placed into the stash. The balanced situation is special because the majority of hosts are low in the same resource. Such scenarios makes the situation closer to a one-dimensional problem, where a smaller VM is easier to place than a larger one. Therefore, `ChooseHostBalanced` (row 12) chooses the destination host h with the greatest number of VMs smaller than v using Eq. (16).

Afterwards, `ForceFitBalanced` (row 13) sorts the VMs in h according to lexicographical order:

1. Prefer VMs previously migrated to h.
2. In case of a tie, prefer VMs with smaller migration cost.

Next, `ForceFitBalanced` iterates over the sorted list of VMs and excludes them from h one by one until there is enough free space in h for v. After placing v into h, the procedure tries to return the excluded VMs which fit into h in reverse order. The remaining VMs are returned as V_e (rows 15).

4.6. Lopsided class

In the Lopsided class (Listing 2 rows 16–19) we also need to eject some VMs from the destination host h into the stash to place a given VM v. However, unlike the balanced case, both resources limit VMs placement. The Lopsided class is a key tool of the BalCon from a balancing point of view. The balance of the whole datacenter improves by the choices of the destination host and VMs to eject for future induced migrations.

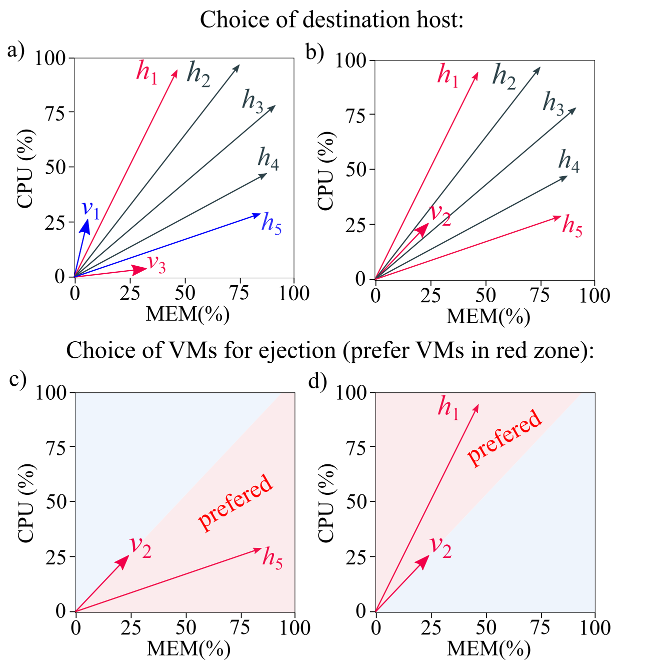
Choice of destination host:
a)
b)
CPU (%)
MEM(%)
h_1
h_2
h_3
h_4
h_5
v_1
v_2
v_3
Choice of VMs for ejection (prefer VMs in red zone):
c)
d)
prefered

Fig. 5. Illustration of choices of destination host and VMs for ejection in the Lopsided class. (a–b) Illustrate the choice of the destination host (h_1, h_5) depending on the VM (v_1, v_2, v_3) to place. In (a) the choice is explicit, and the destination host has the opposite load angle (for v_1 is h_5 and for v_3 is h_1), whereas in (b) the choice is ambiguous and depends on the previous steps (v_2 can be allocated on h_1 and h_5). (c–d) Illustrates choices of VMs for ejection. The preferable VMs are in the red areas that are located on the same side from the VM as the destination host (h_1 or h_5).

Fig. 5 presents an example of the resource requirements of the VMs and the loads of the hosts. Some hosts lack CPU but have plenty of memory (h_1), some hosts lack memory but have plenty of CPU (h_4, h_5), and some are low on both resources. To reduce imbalance, we prefer to place a given VM into the host with minimal or maximal load angles (in example h_1 or h_5).

Definition 7. The load angle of a VM is arctan $\frac{v.cpu}{v.mem}$, and the load angle of a host is arctan $\frac{load(h).cpu}{load(h).mem}$.

To choose the destination host h, the procedure `ChooseHostLopsided` (row 17) uses the global variable $r \in$ {"mem", "cpu"} and follows rules:

1. If the load angle of v is the largest or smallest among the load angles of all the hosts, choose the destination host with the most opposite load angle and set r to the host's largest resource.
   For instance, in Fig. 5a, v_1 has the largest angle and is moved to host h_5, which has the smallest angle (r set to "mem"). Similarly, v_3 has the smallest angle and is moved to h_1, which has the largest angle (r set to "cpu").
2. Else, switch r and choose the host with the largest load by the resource in r.
   For instance, with v_2 from Fig. 5b, if r was "cpu", it switches to "mem", and v_2 is moved to h_5 (host with the largest "mem" load). Otherwise, it switches to "cpu" and is moved to h_1 (host with the largest "cpu" load).

The implementation of the `ForceFitLopsided` procedure (row 18) is similar to that of the Balanced class, except it has an additional requirements on the load angle of the VMs for ejection. To fix the destination hosts lopsidedness, the load angle of the VMs for ejection has to be close to the load angle of the host. The procedure follows lexicographical sorting of the VMs in h:

Table 2
Comparison of time complexity of the algorithms.

Algorithm	Time complexity
SerconOriginal	$O\left(\lvert H\rvert^2 \cdot (V_{max} \cdot \log(V_{max}) + V_{max} \cdot \lvert H\rvert)\right)$
SerconModified	$O\left(\lvert H\rvert \cdot (V_{max} \cdot \log(V_{max}) + V_{max} \cdot \lvert H\rvert)\right)$
BalCon	$O\left(b \cdot \lvert H\rvert \cdot (V_{max} \cdot \log(V_{max}) + V_{max} \cdot \lvert H\rvert)\right)$

Table 3
Statistics of Huawei and synthetic datasets, where mean values are given per instance.

Dataset	Instances total	Hosts mean	VMs mean	Flavors mean/total	Balance factor mean
Huawei	555	96	1.0k	8/69	0.99
Synthetic	200	50	2.7k	28/30	0.07

1. Prefer VMs with a direction to the same side of v as h.
   For instance, if h has a lower angle than v, we prefer VMs with angles less than v (red zone Fig. 5c). Otherwise, we prefer VMs with angle larger than v (red zone Fig. 5d).
2. In case of a tie, prefer VMs which were previously migrated to h.
3. In case of a second tie, prefer VMs with a smaller migration cost.

After sorting, the `ForceFitLopsided` procedure frees space in h for v exactly like in `ForceFitBalanced` and returns the VMs for ejection V_e in row 18.

4.7. Modified sercon heuristics and time complexity

To compare BalCon with well-known approaches, we modified the Sercon heuristic to fit our objectives. The SerconModified heuristic derives from BalCon if we forbid using Force Steps. The differences between the original Sercon heuristic and the SerconModified heuristic include:

1. In Sercon, the authors limited the total number of allowed migrations over all hosts. The SerconModified heuristic limits the amount of migrated memory in each algorithm step. A step is accepted if the new amount of migrated memory in the step is no more than *MPH* (Listing 1 row 10).
2. Unlike Sercon, we try to release each host once.
3. The destination host for a VM in the original version is chosen with First Fit, whereas we use Best Fit. Also, we choose the destination host among all active hosts.
4. We omitted the migration efficiency parameter.

The second modification leads to $|H|$ times less complexity of SerconModified compared to SerconOriginal (see Table 2), where V_{max} is the maximum number of VMs in a host. The complexity of BalCon is only b times worse than that of SerconModified. The upper bounds on V_{max} can be $|V|$.

5. Evaluation

5.1. Datasets

Algorithm evaluation was performed using Huawei and synthetic datasets. Synthetic datasets was generated to test the algorithms under complex inputs, whereas the Huawei datasets validate algorithm performance on real data. The comparison of the main characteristics of the datasets is presented in Table 3 and Fig. 6. In general, synthetic problem instances are less balanced, have more VMs and more flavors than the Huawei datasets.

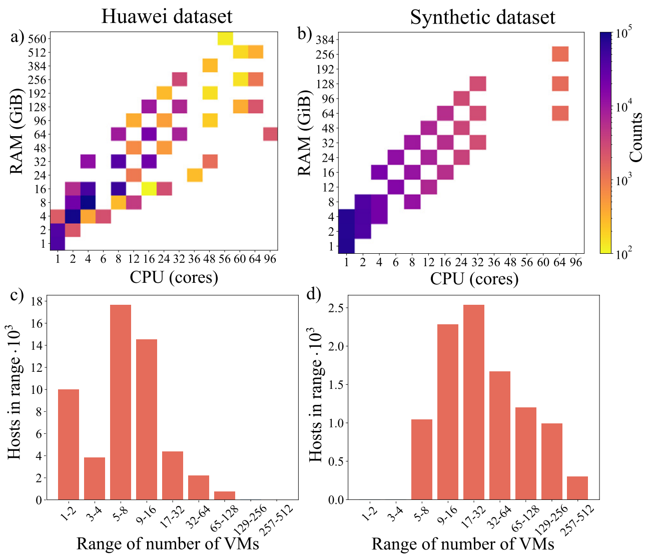

Fig. 6. The comparison between the Huawei (a, c) and synthetic (b, d) datasets. Panels a–b present the flavor distribution over all problem instances, whereas panels (c–d) demonstrate the number of hosts with VMs in the given range.

The Huawei dataset contains 555 cluster snapshots from an operational Huawei Cloud. The snapshots were gathered from clusters of various sizes and roles. The flavor distribution over all snapshots is given in Fig. 6a, whereas the distribution of hosts within a range of the number of VMs is presented in Fig. 6c.

To generate a complex synthetic dateset, we created instances with severe lopsidedness of resources. Briefly, the VMs were generated from 30 flavors with decreasing probabilities by CPU size (see Fig. 6b). Then, the VMs were sorted by the load angle (Definition 7) and packed into hosts using the First Fit heuristic. Such packing method complicates the ability of Sercon-like heuristic because of the lack of free space in hosts. Also, the packing produces an imbalance that requires many Force Steps in BalCon to place VMs with ForceFit. To validate the lopsidedness of resources in our synthetic datasets we calculated the mean balance factor. The balance factor of an instance from the datasets was calculated with Eq. (19), where $s = \frac{1}{|H|}\sum_{h\in H}(h.cpu, h.mem)$ is the mean hosts capacity in the instance. The mean balance factor (Table 3) of the initial feasible mappings in the Huawei instances is 0.99, which corresponds to a balanced situation. In contrast, for the synthetic data, the mean balance factor is 0.07 (Table 3), indicating a highly lopsided situation.

5.2. BalCon performance

We begin evaluation of algorithm performance with the extreme case of the classical VBP problem, where the maximal memory for migration per host (Eq. (2)) is infinite $MPH = \infty$ TiB. To compare BalCon with SerconModified (Section 4.7), we measured the gap between the values of the objective function of the algorithm solution μ_{alg} and the optimal solution μ_{opt}

$$Gap(\mu_{alg}, \mu_{opt}) = \frac{Obj(\mu_{alg}, \mu_0) - Obj(\mu_{opt}, \mu_0)}{Obj(\mu_0, \mu_0) - Obj(\mu_{opt}, \mu_0)}, \tag{25}$$

where μ_{opt} were obtained with the Flavor flow model (Section 3.2). The performance profile [37] indicates the advantage of BalCon over SerconModified, which optimally solved 535 and 345 instances respectively on Huawei dataset (see Fig. 7a). Also, the original version of Sercon (see Section 4.7) optimally solved 245 instances. On the synthetic datasets BalCon optimally solved all the problem instances, whereas Sercon was unable to solve any because of lack of free space of one of the resources (see Fig. 7b and Section 5.1). The iterative nature of the algorithms allows for easy implementation of a running time limit. The time limit for all algorithms was set to 60 s on one physical core of Intel Xeon E5-2690. However, on average, BalCon solved instances in less time (see Table 4). Additionally, the measured execution time is well-correlated with the algorithms time complexity from Table 2.

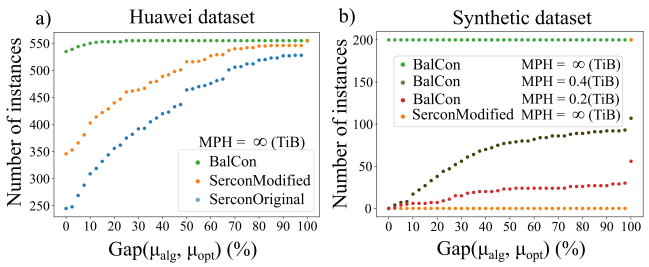

Fig. 7. The performance profiles of BalCon, SerconModified, and SerconOriginal are compared to the optimal solution μ_{opt} of the Flavor flow model for the VBP problem when $MPH = \infty$ (TiB). The Gap is calculated using Eq. (25). Panels (a) and (b) show the comparison for the Huawei and Synthetic datasets, respectively.

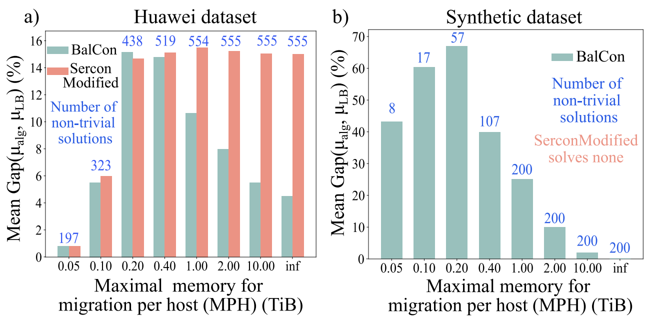

Fig. 8. Comparison of BalCon and SerconModified with lower bound (LB) at different values of MPH. The mean Gap is calculated over non-trivial solutions using Eq. (25). Panels (a) and (b) show the results on the Huawei dataset and Synthetic dataset, respectively.

To investigate the algorithms performance towards migration-aware consolidation, we chose a few values of MPH. Solvers with the Flavor flow model were unable to obtain optimal solutions for all MPH values in a reasonable time. Therefore, we use the LB obtained with the relaxed flavor flow model instead of the true optimum (Section 3.3). In the example of the VBP problem the execution time for the LB is significantly lower (see Table 4). The mean values of $Gap(\mu_{alg}, \mu_{LB})$ over non-trivial solutions – at least one host is released – are presented in Fig. 8. As expected from the VBP results, BalCon outperforms SerconModified on the Huawei dataset and dominates on the Synthetic dataset, where SerconModified is unable to solve any instance. The Force Steps of BalCon allow to provide closer solutions to the LB than SerconModified on the Huawei dataset (Fig. 8a) at large values of MPH from 1 TiB to 10 TiB. However, when MPH reaches the capacities of the hosts $\sim$ 0.7TiB, the number of non-trivial instances decreases, and performance of both algorithms equalizes, due to the insufficient memory for induced migrations. Even at $MPH = 0.2$ TiB, Force Steps lead to a little bit worse performance of BalCon. Qualitatively Balcon demonstrates similar operation in the Synthetic datasets, however the Gap to LB is larger and the number of non-trivial solutions is more sensitive to MPH. The last facts are because of the high imbalance of the Synthetic datasets, which require additional amounts of memory for migration.

Table 4
The comparison of execution time between methods in the case of the VBP problem and Huawei dataset.

Method	ILP flow model	Relaxed ILP flow model (LB)	BalCon	Sercon Original	Sercon Modified
Mean execution time	4 h	0.3 s	9.5 s	0.3 s	0.1 s

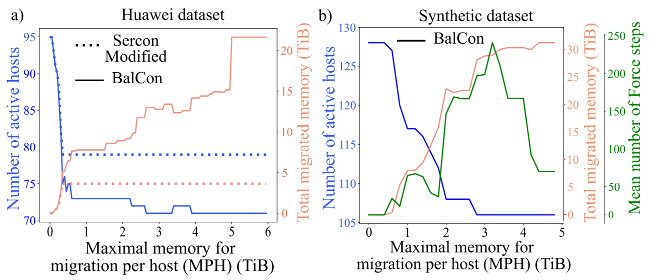

Fig. 9. Examples of BalCon and SerconModified performance at different *MPH* values. The evaluation is demonstrated in terms of the number of active hosts, the amount of migrated memory, and the number of Force Steps. The problem instances are from (a) Huawei dataset and (b) Synthetic dataset.

In general, BalCon's high performance is determined by the Ample, Balanced, and Lopsided classes and their respective heuristics (see Sections 4.4–4.6). Other factors – such as cost functions for VMs ordering – are details of minor importance. For instance, a random ordering of VMs in the stash leads to less than 1% of Gap increase compared to ordering with Eq. (16). To demonstrate the influence of *MPH* on BalCon's performance and areas for further improvement of the algorithm, we selected one representative instance from each dataset (see Fig. 9). The number of active hosts and total migrated memory of SerconModified and BalCon coincide at low *MPH* (Fig. 9a). Then, at larger *MPH*, SerconModified reaches constant values, whereas BalCon releases more hosts at the cost of a larger amount of migrated memory and improved objective function.

Because of the greedy nature, BalCon sometimes performs non-optimal steps by increasing the amount of migrated memory, the number of active hosts, or the number of Force Steps. For instance, the number of active hosts is increased by one in the range of [3.4,3.8] TiB, which could be avoided using the solution at 3 TiB (see Fig. 9a). Also, at the same number of active hosts, the algorithm uses larger memory than it could in the range [5.0, 6.0] TiB because the better solution is at 4 TiB. The mean number of Force Steps changes, although the number of active hosts remains the same for the range [2.8, 3.6] TiB (see Fig. 9b). We would like to emphasize that those are only examples of non-optimal BalCon operations, which might be considered for further algorithm improvement. In general, the number of Force Steps and total migrated memory increase along with *MPH*. Also, on average BalCon demonstrates outstanding performance, especially in the case of large *MPH* and imbalanced situations (Fig. 8).

6. Conclusions

We proposed the BalCon algorithm which efficiently solves the migration-aware consolidation problems. The algorithm was compared with a modified Sercon heuristic and ILP models. The advantages of BalCon over Sercon-like heuristics were achieved due to Force Steps. The Force Steps allow BalCon to optimally solve imbalanced problem instances that Sercon-like heuristics are unable to optimize. The performance of BalCon is very close to optimal at large values of *MPH*. Time complexity of BalCon is only b times larger than that of modified Sercon heuristic, where b is the maximum number of Force Steps. Note that the BalCon implementation is independent of flavor set F and therefore directly applicable to dynamic consolidation. Also, we used the amount of RAM as a migration cost, however one can replace it with other metrics such as number of VM migrations, predicted time for migration of VM, probability of SLA violation, *etc.*

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data and code used in this article are available by the link https://github.com/andreigudkov/BalCon.

Acknowledgments

Authors are grateful to Mr. Oliver Slumbers for helping with language polishing of this article. Dr. Stepan Romanov thanks Dr. Mafuda for fruitful discussions and organizational help.

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, Oper. Syst. Rev. 37 (5) (2003) 164–177.
[2] R. Uhlig, G. Neiger, D. Rodgers, A.L. Santoni, F.C. Martins, A.V. Anderson, S.M. Bennett, A. Kagi, F.H. Leung, L. Smith, Intel virtualization technology, Computer 38 (5) (2005) 48–56.
[3] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al., A view of cloud computing, Commun. ACM 53 (4) (2010) 50–58.
[4] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility, Future Gener. Comput. Syst. 25 (6) (2009) 599–616.
[5] P. Patel, A.H. Ranabahu, A.P. Sheth, Service level agreement in cloud computing, 2009.
[6] G.J. Mirobi, L. Arockiam, Service level agreement in cloud computing: An overview, in: 2015 International Conference on Control, Instrumentation, Communication and Computational Technologies, ICCICCT, IEEE, 2015, pp. 753–758.
[7] A. Beloglazov, J. Abawajy, R. Buyya, Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing, Future Gener. Comput. Syst. 28 (5) (2012) 755–768.
[8] A. Ruprecht, D. Jones, D. Shiraev, G. Harmon, M. Spivak, M. Krebs, M. Baker-Harvey, T. Sanderson, Vm live migration at scale, ACM SIGPLAN Notices 53 (3) (2018) 45–56.
[9] R.W. Ahmad, A. Gani, S.H.A. Hamid, M. Shiraz, A. Yousafzai, F. Xia, A survey on virtual machine migration and server consolidation frameworks for cloud data centers, J. Netw. Comput. Appl. 52 (2015) 11–25.
[10] A. Verma, P. Ahuja, A. Neogi, pMapper: power and migration cost aware application placement in virtualized systems, in: ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing, Springer, 2008, pp. 243–264.
[11] T.C. Ferreto, M.A. Netto, R.N. Calheiros, C.A. De Rose, Server consolidation with migration control for virtualized data centers, Future Gener. Comput. Syst. 27 (8) (2011) 1027–1034.
[12] K.S. Rao, P.S. Thilagam, Heuristics based server consolidation with residual resource defragmentation in cloud data centers, Future Gener. Comput. Syst. 50 (2015) 87–98.
[13] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, J. Lawall, Entropy: a consolidation manager for clusters, in: Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, 2009, pp. 41–50.

[14] A. Murtazaev, S. Oh, Sercon: Server consolidation algorithm using live migration of virtual machines for green computing, IETE Tech. Rev. 28 (3) (2011) 212–231.

[15] Q. Wu, F. Ishikawa, Q. Zhu, Y. Xia, Energy and migration cost-aware dynamic virtual machine consolidation in heterogeneous cloud datacenters, IEEE Trans. Serv. Comput. 12 (4) (2016) 550–563.

[16] Y. Gao, H. Guan, Z. Qi, Y. Hou, L. Liu, A multi-objective ant colony system algorithm for virtual machine placement in cloud computing, J. Comput. System Sci. 79 (8) (2013) 1230–1242.

[17] M. Marzolla, O. Babaoglu, F. Panzieri, Server consolidation in clouds through gossiping, in: 2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, IEEE, 2011, pp. 1–6.

[18] E. Feller, C. Morin, A. Esnault, A case for fully decentralized dynamic VM consolidation in clouds, in: 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings, IEEE, 2012, pp. 26–33.

[19] A. Ashraf, B. Byholm, I. Porres, Distributed virtual machine consolidation: A systematic mapping study, Comp. Sci. Rev. 28 (2018) 118–130.

[20] M.R. Garey, D.S. Johnson, Computers and Intractability, W. H. Freeman, 1979.

[21] E.G. Coffman Jr., M.R. Garey, D.S. Johnson, Bin packing with divisible item sizes, J. Complexity 3 (4) (1987) 406–428.

[22] M.X. Goemans, T. Rothvoss, Polynomiality for bin packing with a constant number of item types, J. ACM 67 (6) (2020) 1–21.

[23] H.I. Christensen, A. Khan, S. Pokutta, P. Tetali, Approximation and online algorithms for multidimensional bin packing: A survey, Comp. Sci. Rev. 24 (2017) 63–79.

[24] R. Aringhieri, D. Duma, A. Grosso, P. Hosteins, Simple but effective heuristics for the 2-constraint bin packing problem, J. Heuristics 24 (3) (2018) 345–357.

[25] R. Panigrahy, K. Talwar, L. Uyeda, U. Wieder, Heuristics for vector bin packing, 2011, research. microsoft. com.

[26] D. Bartók, Z.Á. Mann, A branch-and-bound approach to virtual machine placement, in: Proceedings of the 3rd HPI Cloud Symposium "Operating the Cloud, 2015, pp. 49–63.

[27] B. Speitkamp, M. Bichler, A mathematical programming approach for server consolidation problems in virtualized data centers, IEEE Trans. Serv. Comput. 3 (4) (2010) 266–278.

[28] C. Ghribi, M. Hadji, D. Zeghlache, Energy efficient vm scheduling for cloud data centers: Exact allocation and migration algorithms, in: 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, IEEE, 2013, pp. 671–678.

[29] J. Xu, J.A. Fortes, Multi-objective virtual machine placement in virtualized data center environments, in: 2010 IEEE/ACM Int'L Conference on Green Computing and Communications & Int'L Conference on Cyber, Physical and Social Computing, IEEE, 2010, pp. 179–188.

[30] H. Hallawi, J. Mehnen, H. He, Multi-capacity combinatorial ordering GA in application to cloud resources allocation and efficient virtual machines consolidation, Future Gener. Comput. Syst. 69 (2017) 1–10.

[31] A. Ashraf, I. Porres, Multi-objective dynamic virtual machine consolidation in the cloud using ant colony system, Int. J. Parallel Emergent Distrib. Syst. 33 (1) (2018) 103–120.

[32] H. Li, G. Zhu, C. Cui, H. Tang, Y. Dou, C. He, Energy-efficient migration and consolidation algorithm of virtual machines in data centers for cloud computing, Computing 98 (3) (2016) 303–317.

[33] J. Jiang, Y. Feng, J. Zhao, K. Li, DataABC: A fast ABC based energy-efficient live VM consolidation policy with data-intensive energy evaluation model, Future Gener. Comput. Syst. 74 (2017) 132–141.

[34] R. Ponto, G. Kecskeméti, Z.Á. Mann, Comparison of workload consolidation algorithms for cloud data centers, Concurr. Comput.: Pract. Exper. 33 (9) (2021) e6138.

[35] Y. Ho, P. Liu, J.-J. Wu, Server consolidation algorithms with bounded migration cost and performance guarantees in cloud computing, in: 2011 Fourth IEEE International Conference on Utility and Cloud Computing, IEEE, 2011, pp. 154–161.

[36] J. Forrest, T. Ralphs, H. Santos, et al., coin-or/Cbc: Release releases/2.10. 8, May, 2022.

[37] E.D. Dolan, J.J. Moré, Benchmarking optimization software with performance profiles, Math. Program. 91 (2) (2002) 201–213.

Andrei Gudkov received his M.Sc. degree from the faculty of Computational Mathematics and Cybernetics of Moscow State University, Russia, in 2009. He held multiple engineering positions in hi-tech companies specializing in the areas of full-text search engines, distributed computing and big data. Currently he is employed at Mathematical Modeling Lab of Huawei Moscow Research Center, focusing on resource usage optimization in Huawei Cloud. Possessing deep experience in computing, he models cloud environment at all levels and designs practical scheduling algorithms.

Pavel Popov received his M.S. degree in Mathematics from the National Research University Higher School of Economics, Moscow, in 2015. He has also completed the Yandex School of Data Analysis in 2022. Previously, he conducted research in Algebraic Geometry, focusing on the study of cubic hypersurfaces. Currently, he is working in the field of Combinatorial Optimization, specifically on different Cloud Scheduling Problems. His research aims to develop efficient algorithms for resource allocation in cloud computing systems.

Stepan Romanov holds both a B.S. and an M.S. in computer science from the Moscow Institute of Physics and Technologies. He further pursued his academic passions and obtained an MS and a Ph.D. in physics from the Skolkovo Institute of Science and Technologies. Dr. Romanov is a multidisciplinary scholar with a keen interest in diverse areas such as computational chemistry, experimental physics, robotics, and computer science. Currently, he is actively engaged in researching and tackling challenges in cloud computing systems.