Flow-Aware Workload Migration in Data Centers — Source link

Yoann Desmouceaux, Yoann Desmouceaux, Sonia Toubaline, Thomas Clausen

Institutions: École Polytechnique, Cisco Systems, Inc., Paris Dauphine University

Published on: 01 Oct 2018 - Journal of Network and Systems Management (Springer US)

Topics: Workload, Heuristic (computer science), Linear programming, Network topology and Multi-objective optimization

J Netw Syst Manage manuscript No.
(will be inserted by the editor)

Flow-Aware Workload Migration in Data Centers

Yoann Desmouceaux · Sonia Toubaline · Thomas Clausen

Abstract In data centers, subject to workloads with heterogeneous (and sometimes short) lifetimes, workload migration is a way of attaining a more efficient utilization of the underlying physical machines. To not introduce performance degradation, such workload migration must take into account not only machine resources, and per-task resource requirements, but also application dependencies in terms of network communication.

This paper presents a workload migration model capturing all of these constraints. A linear programming framework is developed allowing accurate representation of per-task resources requirements and inter-task network demands. Using this, a multi-objective problem is formulated to compute a reallocation of tasks that (i) maximizes the total inter-task throughput, while (ii) minimizing the cost incurred by migration and (iii) allocating the maximum number of new tasks.

A baseline algorithm, solving this multi-objective problem using the ε-constraint method is proposed, in order to generate the set of Pareto-optimal solutions. As this algorithm is compute-intensive for large topologies, a heuristic, which computes an approximation of the Pareto front, is then developed, and evaluated on different topologies and with different machine load factors. These evaluations show that the heuristic can provide close-to-optimal solutions, while reducing the solving time by one to two order of magnitudes.

Yoann Desmouceaux*†, E-mail: yoann.desmouceaux@polytechnique.edu, +33158045314 (corresponding author)
Sonia Toubaline‡, E-mail: sonia.toubaline@dauphine.fr, +33144054415
Thomas Clausen*, E-mail: thomas.clausen@polytechnique.edu, +33169334088
*École Polytechnique, 91128 Palaiseau, France
†Cisco Systems Paris Innovation and Research Laboratory (PIRL), 92782 Issy-les-Moulineaux, France
‡Université Paris-Dauphine, PSL Research University, CNRS, LAMSADE, 75016 Paris, France
Neither the entire paper nor any part of its content has been published or has been accepted for publication elsewhere. It has not been submitted to any other journal.

Keywords

Data Center Networking, VM Migration, Application-Aware Allocation, MILP, Multi-objective Optimization, Pareto Optimality.

1 Introduction

1.1 Context and Approach

Virtual Machine (VM) live migration [1] is a way to transfer a VM from a host to another, by iteratively copying its memory. It is further suggested in [2,3] that it is possible to perform live migration of individual processes or containers, thus attaining one further level of granularity of task placement flexibility. In virtualized data centers, applications thus need not be tied to a specific machine – and their relocation can become part of a "natural" mode of data center operation. This provides a way to accommodate hardware failures without service downtime, of course, but more generally may be beneficial to the efficiency of the whole data center in terms of resource usage and execution efficiency.

With the desirability of task (VM or container, also occasionally called *"workload"* in literature [4]) mobility established, the question of an optimal task migration mechanism naturally arises, which will be investigated in this paper. Task placement has been studied, and current approaches provide mathematical models focusing on optimal usage of resources and on flow minimization [5], or models that also consider migration but do not take detailed the network topology into account [6,7].

This paper introduces a multi-objective linear programming model of task migration that satisfies per-node resources constraints and inter-nodes communication requirements while minimizing the cost incurred by migrations. The fundamental assumptions are (i) that tasks have inter-communication demands that change with time, (ii) that tasks that communicate benefit from being topologically close, and therefore (iii) that changes in inter-communication demands may make a current task placement suboptimal, suggesting possible efficiency gains by relocation. Hence, the developed multi-objective linear programming model of task migration is intended to run iteratively, at regular time intervals. It uses task inter-communication demands to decide if and where to reallocate tasks so as to provide the best possible task placement, while minimizing a migration cost. Network constraints are taken into account by modeling the traffic as a multi-commodity flow problem, where a commodity represents an inter-task communication demand.

1.2 Related Work

Data center traffic has been extensively studied: it has been shown that the amount of intra-rack traffic is as voluminous as is extra-rack traffic [8], and that patterns emerge when considering the traffic matrix between interdependent tasks [9,10]. Some task pairs communicate more than others, in which case it makes sense to locate those on either the same machine, or on machines close to each other in the network topology. Therefore, data center architectures have been studied which take communication dependencies between workloads into account, from a network systems perspective [11,12]. This motivates the need for models that can place workloads according to their pairwise communication demands.

The VM placement problem consists of finding an efficient assignment of VMs to machines within a data center, satisfying a set of constraints (resources, possibly traffic demand) with respect to one or more objective functions such as energy consumption minimization [4,13,14,15,16], communication cost minimization [17,5], network traffic minimization or resource utilization maximization. Different optimization approaches have been proposed for the VM placement problem: solution techniques include deterministic algorithms, heuristics, meta-heuristics and approximation algorithms. Surveys of these methods can be found in [18,19,20,21]. Many approaches have been proposed that aim at optimizing a single objective. Multi-objectives formulations have also been introduced, but they are often reduced to a mono-objective problem, failing to provide a set of Pareto-optimal solutions.

Single-objective approaches: In [6], a model that considers resources constraints as well as dependencies between applications is proposed, which then re-allocates VMs while minimizing the migration impact. Contrary to the approach presented in this paper, [6] does not consider changing communication demands between applications, but only migrate VMs located on overloaded machines. In [7], this model is refined by incorporating finer-grained server-side constraints, as well as network capacities of the machines. In [22], the problem of optimizing both VM placement and traffic flow routing so as to save energy is considered. A model is formulated that minimizes the power consumption of switches and links, while satisfying capacity and traffic demands constraints. In [17], a network-aware model is considered that places VMs with the aim of minimizing communication cost. A greedy consolidation algorithm is proposed, consisting of identifying VM clusters based on network traffic (using a cost matrix). In [10], a traffic-aware VM placement model is introduced, where the objective is to minimize the aggregate traffic rates perceived by every switch. A two-tier approach given by a cluster-and-cut heuristic is proposed.

Multi-objective reduced to single-objective: Some VM placement problems are formulated as multi-objectives models, which are then reduced to single-objective models. In [23], the authors consider the placement of VMs on machines by minimizing the size of clusters of VM serving the same jobs, and minimizing the maximum traffic on uplinks of top-of-rack switches. An iterative least-loaded-first based placement algorithm is proposed, which first gives the priority to locality, and then to bandwidth occupancy. In [24], a power-efficient VM placement and migration model is introduced, aiming at minimizing the number of machines used, the network energy consumption and the network end-to-end delay. The model is solved using a weighted sum approach. In [25], VM placement is considered under three objectives, to be minimized: total resource “waste”, power consumption, and thermal dissipation costs. The solution approach is a combination of a generic grouping algorithm and a fuzz multi-objective evaluation. In [26], a VM management framework considering initial VM placement and VM migration is introduced, with three objectives: elimination of thermal hotspots, minimization of power consumption and application performance satisfaction. The problem is solved using a weighted sum approach.

Pure multi-objective approaches: In [27], a bi-objective mathematical formulation for VM placement is proposed, aiming at minimizing resource wastage and power consumption. A multi-objective ant colony system algorithm is proposed to generate the set of non dominated solutions. In [28] (extending [29]), a multi-objective formulation of the VM placement problem is introduced, aiming at minimizing energy consumption and network traffic, and maximizing economical revenue while satisfying a service level agreement (SLA). A memetic algorithm given by an evolutionary process is proposed, and a Pareto set approximation is returned.

1.3 Problem Restatement

For some distributed computing applications, where tasks frequently communicate with each other, the limiting factor for the completion of a running task is not necessarily its actual location (assuming that the task is allocated with the full physical resources it needs), but can be the throughput at which it exchanges data with other tasks. Furthermore, if those tasks have changing traffic requirements, migrating them within the data center can be an efficient way to re-organize the load across all available machines so as to optimally satisfy these requirements.

With these two assumptions as a baseline – tasks need to be allocated according to their communication requirements, and those requirements can change over time – the multi-objective mathematical problem introduced in this paper consists of determining a task placement which satisfies machine, network and applications constraints, while (i) maximizing the number of new tasks to be assigned, (ii) minimizing the migration cost and (iii) maximizing the total inter-task throughput.

The contribution of this paper is therefore threefold: (i) it formulates a model of workload migration which uses an accurate description of network demands and of the network topology; (ii) it proposes a multi-objective approach which generates the set of Pareto-optimal solutions, from which the operator can choose according to operational tradeoffs; and (iii) it proposes a heuristic method that can generate an approximate Pareto front while greatly reducing solving time as compared to obtaining the exact solution.

Evaluations show that the sets of solutions generated by this approximation method remain close to their optimal counterparts, regardless of the topology and the task-to-machine ratio.

1.4 Paper Outline

The remainder of this paper is organized as follows. Section 2 introduces the framework needed to represent the state of the data center. Section 3 presents the mathematical formulation proposed for the flow-aware workload placement and migration problem, given by a multi-objective Mixed Integer Linear Program (MILP) [30]. Section 4 describes the algorithm used to generate the set of Pareto-optimal solutions to this problem, and provides a computational example. An approximation method is proposed in section 5; its performance in terms of solving time and quality of the generated approximate solutions (as compared to the optimal ones) are then evaluated. Finally, section 6 concludes this paper.

2 Data Center Representation

This paper presents a multi-objective time-indexed formulation of a task migration and placement problem. Before detailing the problem formulation (section 3), resolution (section 4) and approximation (section 5), this section introduces the framework used to describe the state (including the topology) of a data center. This is achieved by considering (i) a set of machines, (ii) the network topology connecting them, and (iii) a set of tasks to allocate on these machines. As this model considers workload migration, for which it is necessary to be aware of the evolution of the state of the data center, the model makes the assumption that it is running at a given time t (where t is an abstract index without unit), and that the previous state of the data center corresponds to time $(t-1)$. Thus as a convention, a variable superscripted with t represents the current state of the data center (to be optimized), and a variable superscripted with $(t-1)$ is a known input representing the previous state of the data center. A summary of the notations used throughout this paper is provided in Table 1.

2.1 Tasks and Machines

Tasks to be run at time t are represented by a set I^t. At time t, new tasks can arrive or existing tasks can finish executing. *Arriving tasks* correspond to those in $I^t \setminus I^{t-1}$. *Already existing tasks* correspond to those in $I^t \cap I^{t-1}$, and among those, the ones that were successfully assigned to a machine at time $(t-1)$ have priority over new tasks and *must* be placed to some machine at time t (they can continue execution on the same machine, or be migrated to

Notation	Description
$i, i', \dots \in I^t$	Tasks to be run at time t
$m, m', \dots \in M$	Machines
$s, s', \dots \in S$	Switches
$A \subseteq (M \cup S)^2$	Network edges
$A_{mm'} \in 2^A$	Path $m \mapsto m'$
$c_{uv} > 0$	Capacity of link (u, v)
$\kappa_m > 0$	Machine CPU capacity
$r_i > 0$	Task CPU requirement
$s_i > 0$	Task size
$C^t \subseteq I^t \times I^t$	Communicating tasks at time t
$d^t_{ii'} > 0$	Throughput demand for $i \mapsto i'$ at time t
$x^t_{im} \in \{0, 1\}$	Task i is allocated on machine m at time t
$f^t_{ii'}(u, v) \in \mathbb{R}^+$	Flow for $i \mapsto i'$ along (u, v) at time t

Table 1 Inputs and variables to the model

another one, but not be stopped). *Terminated tasks* correspond to those in $I^{t-1} \setminus I^t$, and thus are not of interest to the model.

Machines are represented by a set M. Each machine $m \in M$ has a *CPU capacity* $\kappa_m > 0$ which represents the amount of work it can accommodate. Conversely, each task $i \in I^t$ has a *CPU requirement* $r_i > 0$, representing the amount of resources it needs in order to run. Finally, each task $i \in I^t$ has a *size*[1] $s_i > 0$. This size will be used to model the cost of migrating the task from one machine to another.

2.2 Network Model

In order to take application dependencies into account, the physical network topology existing between the machines must be known. To that end, a network is modeled as a directed graph $G = (V, A)$, where $V = M \cup S$ is the set of vertices, with S the set of switches[2] and A the set of arcs. An arc $(u, v) \in A$ can exist between two switches or between a machine and a switch, but also between a machine and itself to model a loopback interface (so that two tasks on the same machine can communicate). Each of those arcs represents a link in the network, and has a capacity of $c_{uv} > 0$. For each ordered pair of machines $(m, m') \in M^2$, a list $A_{mm'} \in 2^A$ represents the path from m to m'.

For example, given the topology depicted in Figure 1 with three switches s_0, s_1, s_2 and four machines m_1, m_2, m_3, m_4, the arcs of the graph will be as follows: $A = \{(m_1, m_1), (m_1, s_1), (s_1, m_1), (m_2, m_2), (m_2, s_1), (s_1, m_2), (m_3, m_3), (m_3, s_2), (s_2, m_3), (m_4, m_4), (m_4, s_2), (s_2, m_4), (s_1, s_0), (s_0, s_1), (s_2, s_0), (s_0, s_2)\}$. The path from *e.g.,* m_1 to m_3 will be $A_{m_1 m_3} = \{(m_1, s_1), (s_1, s_0), (s_0, s_2), (s_2, m_3)\}$.

[1]For instance, the size of RAM plus storage for a virtual machine.

[2]This term is used generically to refer to any forwarding node in the network, regardless of its actually being a router or a switch.

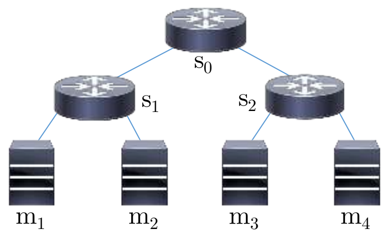

Fig. 1 Simple network example

Finally, at a given time t, an ordered pair of tasks $(i, i') \in I^t \times I^t$ can communicate with a *throughput demand* $d^t_{ii'} > 0$, representing the throughput at which i would like to send data to i'. Let $G^t_d = (I^t, C^t)$ be a weighted directed graph representing these communication demands, where each arc $(i, i') \in C^t$ is weighted by $d^t_{ii'}$. G^t_d will be referred to as the *throughput demand graph.*

3 Mathematical Modeling

Based on the data center framework described in section 2, this section presents a multi-objective Mixed Integer Non Linear Program (MINLP) aiming at optimizing workload allocation and migration while satisfying inter-application network demands. A linearization as a multi-objective MILP is then derived, allowing for an easier resolution.

3.1 Variables

Two sets of variables are introduced, representing (i) task placement and (ii) the network flow for a particular allocation.

Task placement The aim of the model is to provide a placement of each task $i \in I^t$ on a machine $m \in M$, at a given timestep t. The binary variable x^t_{im} reflects this allocation: $x^t_{im} = 1$ if i is placed on m, and $x^t_{im} = 0$ otherwise.

Network flow In order to determine the best throughput that can be achieved between each pair of communicating tasks, a variant of the *multi-commodity flow problem* [31, p. 58] is used, where a commodity is defined by the existence of $(i, i') \in C^t$. A commodity thus represents an inter-task communication.

For each link $(u, v) \in A$, $f^t_{ii'}(u, v)$ is a variable representing the throughput for communication from i to i' along the link (u, v).

3.2 Constraints

Two sets of constraints are used to model this flow-aware workload migration problem: *allocation constraints* (equations (1-3)) represent task allocation, whereas *flow constraints* (equations (4-8)) focus on network flow computation.

The *allocation constraints* represent the relationship between tasks and machines. First, each task $i \in I^t$ must be placed on at most one machine:

$$\sum_{m \in M} x^t_{im} \leq 1, \forall i \in I^t \tag{1}$$

Forcefully terminating a task is not desirable: if a task was running at time $t-1$, and is still part of the set of tasks at time t, it must not be forcefully terminated:

$$\sum_{m \in M} x^{t-1}_{im} \leq \sum_{m \in M} x^t_{im}, \forall i \in I^{t-1} \cap I^t \tag{2}$$

where x^{t-1}_{im} is a known input given by the state of the system at time $t-1$. If the task was successfully allocated at time $t-1$ (*i.e.* $i \in I^{t-1}$ and $\sum_{m \in M} x^{t-1}_{im} = 1$) and is still to be run at time t (*i.e.* $i \in I^t$), the left-hand-side of the equation will be 1, thus forcing allocation of the task at time t.

Finally, the tasks on a machine cannot use more CPU resources than the capacity of that machine:

$$\sum_{i \in I^t} r_i \, x^t_{im} \leq \kappa_m, \forall m \in M \tag{3}$$

The *flow constraints* allow computing the throughput for each commodity (*i.e.* ordered pair of communicating task). For each link $(u, v) \in A$ in the network, the total flow along the link must not exceed its capacity:

$$\sum_{(i,i') \in C^t} f^t_{ii'}(u, v) \leq c_{uv}, \forall (u, v) \in A \tag{4}$$

For a commodity $(i, i') \in C^t$, the flow going out of a machine m must not exceed the *throughput demand* for the communication from i to i'. Also, the flow must be zero if task i is not hosted by machine m:

$$\sum_{v:(m,v) \in A} f^t_{ii'}(m, v) \leq d^t_{ii'} \, x^t_{im}, \forall m \in M, \forall (i, i') \in C^t \tag{5}$$

Conversely, for a commodity $(i, i') \in C^t$, the flow entering a machine m' must not exceed the *throughput demand* for the communication from i to i', and must be set to zero if task i' is not on m':

$$\sum_{v:(v,m') \in A} f^t_{ii'}(v, m') \leq d^t_{ii'} \, x^t_{i'm'}, \forall m' \in M, \forall (i, i') \in C^t \tag{6}$$

Each switch $s \in S$ must forward the flow for each commodity – that is, the ingress flow must be equal to the egress flow:

$$\sum_{v:(u,v)\in A} f^t_{ii'}(u,v) = \sum_{v:(v,u)\in A} f^t_{ii'}(v,u), \forall u \in S, \forall (i,i') \in C^t \tag{7}$$

Finally, if a task i is placed on machine m and a task i' on machine m', the corresponding flow must go through the path specified by $A_{mm'}$. Otherwise, the flow computed by the model could go through a non-optimal path or take multiple parallel paths, which does not accurately reflects what happens in an IP network. Hence, the flow needs to be set to zero for all edges that do not belong to the path from m to m':

$$\begin{aligned} f^t_{ii'}(u,v) \le c_{uv}(1 - x^t_{im}\, x^t_{i'm'}), \\ \forall (i,i') \in C^t, \forall m,m' \in M, \forall (u,v) \in A \setminus A_{mm'} \end{aligned} \tag{8}$$

This constraint has no side effect if task i is not on m or task i' is not on m', since in this case it reduces to $f^t_{ii'}(u,v) \le c_{uv}$, already covered by equation (4).

3.3 Objective Functions

The migration model as presented in this paper introduces three different objective functions, modeling (i) the placement of tasks, (ii) the overall throughput achieved in the network and (iii) the cost incurred by task migration, respectively. These functions depend on an allocation, *i.e.* on an assignment of all variables x^t_{im} and $f^t_{ii'}(u,v)$. Let $\mathbf{x}^t$ (respectively $\mathbf{f}^t$) be the characteristic vectors of the variables x^t_{im} (respectively $f^t_{ii'}(u,v)$).

The *placement objective* is simple and expresses that a maximal number of tasks should be allocated. When removing tasks dependencies, this degenerates to a standard assignment problem wherein each task should be placed on a machine satisfying its CPU requirement, while also not exceeding the machine capacity. The placement objective function is simply the number of tasks which are successfully allocated to a machine:

$$P(\mathbf{x}^t) = \sum_{i\in I^t} \sum_{m\in M} x^t_{im} \tag{9}$$

The *throughput objective* expresses the need to satisfy applications *throughput demands.* Modeling network dependencies between workloads in a data center is usually (*e.g.,* in [6]) done through the use of a cost function depending on the network distance between two machines. Having introduced a representation of the physical network and of applications dependencies in section 2.2, it is possible to represent the overall throughput reached in the data center. To compute the throughput of the communication from a task i to a task i', it suffices to identify the machine m on which i is running and take the flow going out of this machine for this commodity: $\sum_{m\in M} x^t_{im} \sum_{v:(m,v)\in A} f^t_{ii'}(m,v)$. This expression is quadratic in variables x^t_{im} and $f^t_{ii'}(u,v)$, but, owing to the fact that (5) constrains the flow to be zero for machines to which i is not assigned, can be simplified to $\sum_{m\in M} \sum_{v:(m,v)\in A} f^t_{ii'}(m,v)$. Therefore, the overall throughput in the data center can be expressed as:

$$T(\mathbf{f}^t) = \sum_{(i,i')\in C^t} \sum_{m\in M} \sum_{v:(m,v)\in A} f^t_{ii'}(m,v) \tag{10}$$

Since each machine has a loopback link in the graph G (*i.e.* that $(m,m) \in A, \forall m \in M$), this formulation also covers the case where i and i' are hosted on the same machine.

Finally, the *migration cost* reflects the cost of re-allocating tasks from one machine to another. The fundamental assumption is that tasks in a data center have communication demands that can evolve over time (modeled by $d^t_{ii'}$). This means that migrating a task to a machine topologically closer to those machines hosting other tasks with which it communicates, can be a simple way to achieve overall better performance. The migration cost incurred between two successive times $(t-1)$ and t is modeled as the sum of the sizes of tasks that have migrated. Since at time t, the assignment $\mathbf{x}^{t-1}$ is known, it is possible to know if a task $i \in I^t \cap I^{t-1}$ placed on a machine $m \in M$ has moved, by comparing x^t_{im} to $(1 - x^{t-1}_{im})$. Also, if a task is to be shut off at time t (*i.e.* $i \in I^{t-1} \setminus I^t$), it must not be part of the computation of the number of migrated tasks. The total migration cost from time $t-1$ to time t can thus be expressed as:

$$M(\mathbf{x}^t) = \sum_{i\in I^t\cap I^{t-1}} \sum_{m\in M} s_i\, x^{t-1}_{im}(1 - x^t_{im}) \tag{11}$$

Using these three objectives, it is possible to express the flow-aware placement and migration model as a multi-objective MILNP:

$$(\mathcal{P}) \begin{cases} \max T(\mathbf{f}^t) \\ \max P(\mathbf{x}^t) \\ \min M(\mathbf{x}^t) \\ \text{subject to} \begin{cases} \text{(1-8)} \\ \mathbf{x}^t \in \{0,1\}^{I^t\times M} \\ \mathbf{f}^t \in (\mathbb{R}^+)^{C^t\times A} \end{cases} \end{cases} \tag{12}$$

These objectives tend to compete with each other. Indeed, if a task starts communicating with another task, migrating it to the same machine (or to a machine closer in the topology) can increase the throughput at which they communicate, thus increasing the *throughput objective* function – but this will incur an increase of the *migration cost.* Likewise, if a new task arrives at t and needs important CPU resources, placing it right away could utilize CPU capacity that could have been used for co-locating tasks: increasing the *placement objective* can in some cases decrease the *throughput objective*.

As an illustration, consider two machines m_1 and m_2 with CPU capacities $\kappa_1 = 1$ and $\kappa_2 = 2$, with a bottleneck link (of capacity 1) in between. Assume that at $(t-1)$ there were two tasks i_1, i_2 with CPU requirements $r_{i_1} = 1$ and $r_{i_2} = 1$, and that i_1 was placed on m_1 and i_2 on m_2. At time t, i_1 starts to communicate with i_2 (with throughput demand $d^t_{i_1 i_2} = 2$), and a third task i_3 arrives with CPU requirement $r_{i_3} = 2$. Then, if one decides to place this new task, m_2 is the only machine with enough capacity to host it. But placing i_3 on m_2 would prevent a migration of i_1 from m_1 to m_2, which would have increased the throughput of communication $i_1 \mapsto i_2$. This situation is summarized in figure 2.

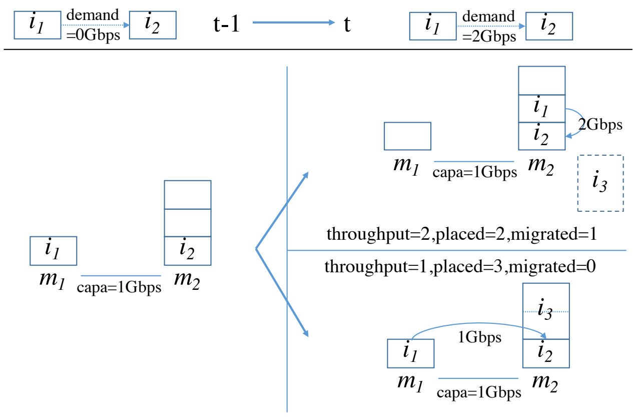

Fig. 2 Competition between objective functions

3.4 Linearization

All constraints expressed in section 3.2 are linear with respect to the variables x^t_{im} and $f^t_{ii'}(u,v)$, except for the *path enforcement* constraint in equation (8). Exploiting the fact that $x^t_{im} \in \{0,1\}$, this constraint can be linearized:

$$f^t_{ii'}(u,v) \le c_{uv}(2 - x^t_{im} - x^t_{i'm'}),$$
$$\forall (i,i') \in C^t, \forall m, m' \in M, \forall (u,v) \in A \setminus A_{mm'} \qquad (13)$$

If $x^t_{im} \times x^t_{i'm'} \neq 1$, the equation will be $f^t_{ii'}(u,v) \le c_{uv}$ or $f^t_{ii'}(u,v) \le 2\ c_{uv}$, and will therefore be superseded by equation (4).

The set of constraints can be further compressed by writing only one equation per machine $m \in M$ instead of one per tuple $m, m' \in M$. This does not alter the model but makes the formulation more compact:

$$f^t_{ii'}(u,v) \le c_{uv} \left(2 - x^t_{im} - \sum_{m' \in M : (u,v) \notin A_{mm'}} x^t_{i'm'} \right), \quad \forall (i,i') \in C^t, \forall m \in M, \forall (u,v) \in A \tag{14}$$

In this way, the flow-aware workload migration problem can be expressed as the following multi-objective Mixed Integer Liner Program (MILP):

$$(\mathcal{P}') \begin{cases} \max T(\mathbf{f}^t) \\ \max P(\mathbf{x}^t) \\ \min M(\mathbf{x}^t) \\ \text{subject to} \begin{cases} \text{(1-7), (14)} \\ \mathbf{x}^t \in \{0,1\}^{I^t \times M} \\ \mathbf{f}^t \in (\mathbb{R}^+)^{C^t \times A} \end{cases} \end{cases} \tag{15}$$

4 Flow-Aware Workload Migration

4.1 Resolution Algorithm

This section describes an algorithm (Algorithm 1) solving the previously introduced multi-objective MILP (15). This algorithm runs at a given time t representing the “present”. It takes the current inter-application communication requirements and the “past” (from time $t-1$) task allocation as inputs, and returning a new allocation as a solution.

A Pareto-optimal approach [30, p. 24] is used: a set of optimal solutions is generated, from which a final solution $\mathbf{z}^t = (\mathbf{x}^t, \mathbf{f}^t)$ can be extracted based on an operator-defined policy *getSolutionFromPolicy()*. In the context of this migration model, an allocation $\mathbf{z} = (\mathbf{x}, \mathbf{f})$ dominates an allocation $\mathbf{z}' = (\mathbf{x}', \mathbf{f}')$ if all the objectives functions of the former have better values than the ones of the latter (with at least of one having a strictly better value). That is denoted $\mathbf{z} \succ \mathbf{z}'$:

$$\mathbf{z} \succ \mathbf{z}' \Leftrightarrow \begin{cases} P(\mathbf{z}) \ge P(\mathbf{z}') \\ T(\mathbf{z}) \ge T(\mathbf{z}') \\ M(\mathbf{z}) \le M(\mathbf{z}') \\ (P(\mathbf{z}), T(\mathbf{z}), M(\mathbf{z})) \ne (P(\mathbf{z}'), T(\mathbf{z}'), M(\mathbf{z}')) \end{cases} \tag{16}$$

A set $\mathcal{S} = \{\mathbf{z}_1, \ldots, \mathbf{z}_n\}$ of solutions is Pareto-optimal if and only if:

$$\forall \mathbf{z}, \mathbf{z}' \in \mathcal{S}, \mathbf{z} \ne \mathbf{z}' \Rightarrow (\mathbf{z} \not\succ \mathbf{z}' \land \mathbf{z}' \not\succ \mathbf{z}) \tag{17}$$

To generate the set of Pareto-optimal solutions for the multi-objective problem, the ε-constraint method [30, p. 98] is used. It consists of reducing

Algorithm 1 Multi-Objective Migration Algorithm

```
$\mathbf{z}^{t-1} \leftarrow$ input    ▷ initial state
$C^t, d^t_{ii'}, B, \underline{\varepsilon_p}, \overline{\varepsilon_p} \leftarrow$ input    ▷ parameters
$\mathcal{S} \leftarrow \emptyset$
for $\varepsilon_p \in [\underline{\varepsilon_p}, \overline{\varepsilon_p}]$ with step $\delta_p$ do
    for $\varepsilon_m \in [B, 0]$ with step $-\delta_m$ do
        if $\exists \mathbf{z}' \in \mathcal{S}, P(\mathbf{z}') \geq \varepsilon_p \wedge M(\mathbf{z}') \leq \varepsilon_m$ then
            continue    ▷ dominant solution already existing
        end if
        $\mathbf{z} \leftarrow$ solution of (18)
        ▷ if $\mathbf{z}$ is not dominated, it is a candidate solution
        if $\nexists \mathbf{z}' \in \mathcal{S}, \mathbf{z}' \succ \mathbf{z}$ then
            ▷ remove any existing solution dominated by $\mathbf{z}$
            for $\mathbf{z}' \in \mathcal{S}$ do
                if $\mathbf{z} \succ \mathbf{z}'$ then
                    $\mathcal{S} \leftarrow \mathcal{S} \setminus \{\mathbf{z}\}$
                end if
            end for
            $\mathcal{S} \leftarrow S \cup \{\mathbf{z}\}$    ▷ add $\mathbf{z}$ to candidate solutions
        end if
    end for
end for
▷ select the best solution according to the considered policy
$\mathbf{z}^t \leftarrow$ getSolutionFromPolicy($\mathcal{S}$)
```

a multi-objective linear programming model to a single-objective model, by relaxing all objective functions but one and considering them as constraints. These constraints are obtained by bounding the relaxed objective function by a number ε that varies in a certain interval. Although its initial formulation only considers two-objectives problems, it can be extended to problems with a greater number of objective functions [32], as it is the case here.

The two functions that are relaxed are the *placement objective* and the *migration cost*, because it is simple to grid the corresponding space. The single-objective MILP that results is given by:

$$(\mathcal{P}_\varepsilon) \begin{cases} \max T(\mathbf{f}^t) \\ \text{subject to} \begin{cases} \text{(1-7), (14)} \\ P(\mathbf{x}^t) \geq \varepsilon_p \\ M(\mathbf{x}^t) \leq \varepsilon_m \\ \mathbf{x}^t \in \{0,1\}^{I^t \times M} \\ \mathbf{f}^t \in (\mathbb{R}^+)^{C^t \times A} \end{cases} \end{cases} \tag{18}$$

where $\varepsilon_p, \varepsilon_m$ are bounds varying in $[\underline{\varepsilon_p}, \overline{\varepsilon_p}]$ and $[0, B]$ respectively.

For the *placement objective*, a simple upper bound for ε_p corresponds to the number of tasks:

$$\overline{\varepsilon_p} = \sum_{i \in I^t} 1 = |I^t| \tag{19}$$

Similarly, a lower bound for ε_p can be easily derived from tasks that were running at the previous iteration and that are still to be run. As per equation

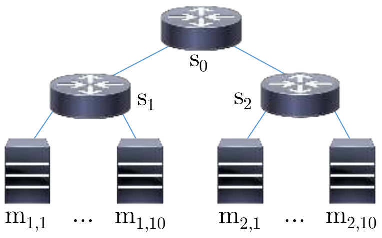

Fig. 3 Network topology used in the example run

$u = s_0, v \in \{s_1, s_2\}$	$c_{uv} = c_{vu} = 40$
$u \in \{s_1, s_2\}, v \in M$	$c_{uv} = c_{vu} = 40$
$u \in M$	$c_{uu} = 200$

Table 2 Link capacities for the example topology

(2), they cannot be forcefully terminated:

$$\underline{\varepsilon_p} = \sum_{i \in I^{t-1} \cap I^t} \sum_{m \in M} x_{im}^{t-1} \tag{20}$$

For the *migration cost*, correctly choosing the upper bound B for ε_m has a large impact on the running time of the algorithm. This bound B will be referred to as the *migration budget*.

4.2 Computational Example

In large-scale deployments, the amount of CPU resources is likely sufficient for all tasks to be deployed. However, complex situations can exist, for which it is beneficial to have an understanding of the implications that different allocation choices can have. For instance, when new tasks arrive that have a high CPU requirement, choosing to place them upfront can consume CPU capacity which could have been used to migrate a smaller task and satisfy its *throughput demand*. This section illustrates the model response in situations where placement and migration are competing with each other, by way of an environment in which the machines are nearly overloaded in the initial state. The algorithm has been implemented in Python using the Gurobi MILP solver [33], and all the simulations have been run on a virtual machine with 4 GB of RAM and an 2.6 GHz CPU.

4.2.1 Simulation Parameters

A simple two-tier tree topology is used, consisting of one root switch s_0 linked to two ToR switches s_1, s_2, each linked to 10 machines (Figure 3). In order to represent the fact that tasks located on the same machine can achieve a

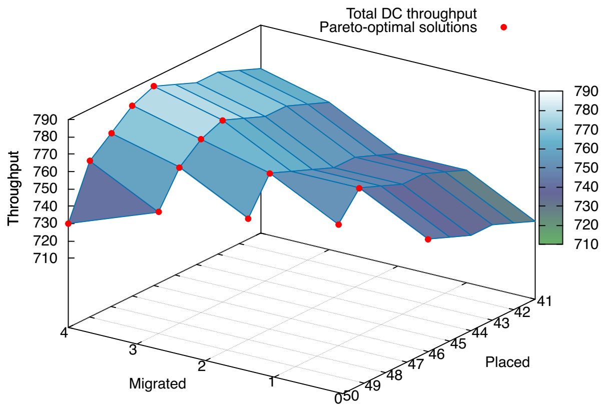

Fig. 4 Multi-objective model example run: solutions and Pareto-optimal solutions.

higher throughput, each machine has a link to itself, with a higher capacity than other links in the data center, as depicted in Table 2.

The CPU capacity is set to 3 for each machine – which gives a total capacity of 60. An initial state with 40 tasks of size $r_i = 1$ and a *throughput demand* graph G_d^{t-1} with 80 arcs of intensity $d_{ii'}^{t-1} = 10$ is considered, with an initial allocation $\mathbf{z}^{t-1}$ optimal with respect to *throughput demands*[3].

Given this allocation, this study explores the model behavior when communication demands of existing tasks change, while new bigger tasks arrive in the system. Communication demand changes are modeled by adding 10 random arcs (of intensity $d_{ii'}^{t} = 10$) between the already running tasks. Arrival of new tasks is modeled by creating 10 new tasks of size $r_i = 2$, and by adding 10 random arcs (of intensity $d_{ii'}^{t} = 5$) going out of these tasks.

4.2.2 Results

In order to obtain the set of Pareto-optimal solutions $\mathcal{S}$ at time t, one iteration of Algorithm 1 was run. Figure 4 shows a three-dimensional representation of the solutions computed for one of those runs: Pareto-optimal solutions can be visually identified as those for which no other solutions have better coordinates in all dimensions. Results show that non-trivial behavior occurs when the data center is almost fully loaded: while placing a few new tasks improves the overall throughput, allocating all of them has a detrimental effect on the overall throughput because it prevents migration of other tasks.

This shows the interest of using a multi-objective model in cases where the objectives are conflicting with each other.

[3]$\mathbf{z}^{t-1}$ is obtained by starting from a random state and solving (15) where only the throughput objective is considered, with the additional constraint that all tasks must be allocated.

Algorithm 2 Heuristic Multi-Objective Resolution

```
$\mathbf{z}^{t-1} \leftarrow$ **input**    $\triangleright$ *initial state*
$C^t, d^t_{ii'}, B \leftarrow$ **input**    $\triangleright$ *parameters*
$m \leftarrow 0$    $\triangleright$ *total migration cost*
$\Delta \leftarrow$ **input**    $\triangleright$ *incremental budget*
$\hat{\mathcal{S}} \leftarrow \emptyset$
$\hat{\mathbf{z}}^0 \leftarrow \mathbf{z}^{t-1}$
$\tau \leftarrow 1$    $\triangleright$ *step index*
**while** $m < B$ **do**
    $\triangleright$ *reduce $\Delta$ at last iteration so that $m + \Delta \leq B$*
    $\Delta \leftarrow \min\{\Delta, B - m\}$
    $\triangleright$ *find an allocation within $\Delta$ of the previous one*
    $\hat{\mathbf{z}}^\tau \leftarrow$ solution of $\begin{cases} \max T(\hat{\mathbf{f}}^\tau) \\ \text{subject to} \begin{cases} \text{(1-7), (14) with } t \mapsto \tau \\ P(\hat{\mathbf{x}}^\tau) = |I| \\ M(\hat{\mathbf{x}}^\tau, \hat{\mathbf{x}}^{\tau-1}) \leq \Delta \\ \hat{\mathbf{x}}^\tau \in \{0,1\}^{I \times M} \\ \hat{\mathbf{f}}^\tau \in (\mathbb{R}^+)^{C^t \times A} \end{cases} \end{cases}$
    **if** $M(\hat{\mathbf{x}}^\tau, \hat{\mathbf{x}}^{\tau-1}) = 0$ **then**
        **break**    $\triangleright$ *stop if no task could be migrated*
    **end if**
    $\hat{\mathcal{S}} \leftarrow \hat{\mathcal{S}} \cup \{\hat{\mathbf{z}}^\tau\}$
    $\triangleright$ *compute the migration cost w.r.t. initial state*
    $m \leftarrow M(\hat{\mathbf{x}}^\tau, \hat{\mathbf{x}}^0)$
    $\tau \leftarrow \tau + 1$
**end while**
$\mathbf{z}^t \leftarrow$ *getSolutionFromPolicy*$(\hat{\mathcal{S}})$
**return** $\mathbf{z}^t$
```

5 Pareto Front Approximation

Section 4 introduced a generic solution approach, which allows to compute the exact Pareto front for the three competing objectives (placing the highest number of tasks, minimizing the cost incurred by migration, achieving the best inter-application throughput) by means of an ε-constraint method. Although this approach generates an exact set of efficient solutions, and thus can be useful for small and highly-constrained deployments, its computational complexity makes it unsuitable to model larger infrastructures.

In order to overcome this issue, this section derives and experimentally analyses a heuristic for approximating the Pareto front. Section 5.1 introduces the approximation method, and section 5.2 provides the results of computational experiments in order to validate the quality of the proposed approach.

5.1 Heuristic Formulation

In large deployments, it is assumed that there are enough resources to accommodate all tasks to be run, thus the remainder of this section assumes that all tasks can be placed (*i.e.* that there exists at least one solution to (18) such

$u = s_0, v \in \{s_1, \ldots, s_k\}$	$c_{uv} = c_{vu} = 40$
$u \in \{s_1, \ldots, s_k\}, v \in M$	$c_{uv} = c_{vu} = 40$
$u \in M$	$c_{uu} = 1000$

Table 3 Link capacities for the *k-racks topology*

that $P(\mathbf{x}) = |I^t|$). In order to focus on migration, it is also assumed that no new task arrives (*i.e.* $I^t = I^{t-1} := I$).

Given this assumption, a heuristic allowing to approximate the Pareto front in the *throughput-migration plane*, *i.e.* for $(P(\mathbf{z}), T(\mathbf{z}), M(\mathbf{z})) \in \{|I|\} \times [0, +\infty) \times [0, B]$, is introduced. The issue with the ε-constraint approach introduced in section 4 is that, when considering the migration of B tasks, there are $\mathcal{O}(|I|^B |M|^B)$ possible recombinations, which can be computationally expensive. In order to limit the combinatorics of the recombination, the introduced heuristic splits the migration process in smaller chunks, considering fewer migrations at a time, and proceeding in a greedy way until the total budget is exhausted, as described in Algorithm 2.

For clarity, let $M(\mathbf{x}^t, \mathbf{x}^{t'})$ represent the migration cost when going from state $\mathbf{x}^t$ to $\mathbf{x}^{t'}$:

$$M(\mathbf{x}^t, \mathbf{x}^{t'}) = \sum_{i \in I \cap I^{t'}} \sum_{m \in M} s_i \, x_{im}^{t'} (1 - x_{im}^t) \tag{21}$$

Let $\Delta > 0$ be a parameter representing an *incremental budget*. Starting from an initial configuration $\mathbf{z}^{t-1}$, and given an upper-bound B on the migration cost, intermediary states $(\hat{\mathbf{z}}^{\mathbf{0}} = \mathbf{z}^{t-1}, \hat{\mathbf{z}}^{\mathbf{1}}, \ldots, \hat{\mathbf{z}}^{\tau}, \ldots, \hat{\mathbf{z}}^{\overline{\tau}})$ are created. At each step $\tau \geq 1$, a new placement is generated by maximizing the throughput objective, with the constraint that no more than Δ is incurred as a migration cost as compared to step $(\tau - 1)$. This way, no more than $\mathcal{O}(|I|^{\Delta} |M|^{\Delta})$ recombinations are considered at each step, allowing for a smaller expected computational complexity. This process ends when no further tasks can be migrated, or when a total migration cost of B (as compared to step 0) has incurred. The heuristic tracks of the solutions generated while generating the successive solutions, so as to build an approximate Pareto front $\hat{\mathcal{S}}$.

5.2 Computational Experiments

This section experimentally illustrates the benefits of using the Pareto front approximation heuristic, both in terms of computational complexity and quality of the approximation. The sensitivity of the heuristic to the *incremental budget* parameter Δ is also experimentally explored. The experimental platform is the same as in section 4.2.

5.2.1 Simulation Parameters

In a common data center, all machines in a rack are linked to a top-of-rack (ToR) switch, and those ToR switches are linked to each other via an aggregation router. In data centers, several such configurations can be linked together thanks to a third routing layer – in all cases, resulting in a tree-like topology.

For the purposes of these simulations, a *k-racks topology* is defined as a tree where a core switch s_0 is attached to k ToR switches $s_1, s_2, \dots, s_k$. Each ToR switch s_h $(1 \leq h \leq k)$ is attached to 10 machines $m_{h,1}, m_{h,2}, \dots, m_{h,10}$. This topology thus comprises $10 \times k$ machines, and Figure 3 depicts a *2-racks topology*. Links capacities are set as per Table 3, *i.e.* links between the core switch and the ToR switches, and between the ToR switches and the machines have a capacity of 40. Loopback interfaces on the machines have a capacity of 1000 – so as to favor placement of tasks which communicate with each other on the same machine.

Five *k-racks topologies* are considered, for $k \in \{2, 3, 4, 5, 10\}$ (for a corresponding number of machines $|M| \in \{20, 30, 40, 50, 100\}$). For each of these topologies, four scenarios are considered with different tasks-per-machine ratios, such that $|I| \in \{|M|, 2|M|, 5|M|, 10|M|\}$ – as a way to compare between different load factors for the machines. This makes for a total of $5 \times 4 = 20$ scenarios.

CPU requirements of tasks are set to $r_i = 1$, and CPU capacities of the machines to $\kappa_m = 20$. This allows for up to 20 tasks per machine, thus ensuring that all tasks can always be placed. To simplify understanding the migration cost, tasks size (for computing the migration cost) are set to $s_i = 1$, thus the migration cost simply becomes the number of migrated tasks. Tasks are initially uniformly randomly assigned to the machines. The communication matrix is generated by: (i) for each task $i \in I$, uniformly choosing a random number d_i of destinations in $\{1, 2, 3\}$, (ii) for each of these d_i destinations, uniformly choosing a task $i' \in I \setminus \{i\}$ and (iii) setting a throughput demand $d_{ii'}$ for the communication $i \mapsto i'$ uniformly in $\{1, 2, \dots, 10\}$.

For each of these 20 scenarios, the exact Pareto front is computed using Algorithm 1, with a migration budget (maximum migration cost) $B = 30$. The approximate Pareto front as described in Algorithm 2 is computed for five different values of the *incremental budget* $\varDelta \in \{1, 2, 3, 4, 5\}$.

5.2.2 Comparative Results

In order to illustrate the results, the cases of the *2-racks topology* (20 machines) with 40 and 100 tasks are first considered as examples. For these two cases, Figures 5 and 6 depict the generated exact and approximate Pareto fronts. Those figures show the different solutions from among which one can be choose, after running either algorithm. As depicted in the figures, migrating more tasks obviously increases the migration cost – while enabling communicating tasks to be co-located, hence increasing the total achievable throughput.

Depending on requirements, a solution can be chosen from the Pareto-optimal set that offers a tradeoff between improving the overall throughput and incurring more migrations. In these two examples, it can be observed that the approximate Pareto fronts generated by algorithm 2 for $\varDelta \geq 3$ closely

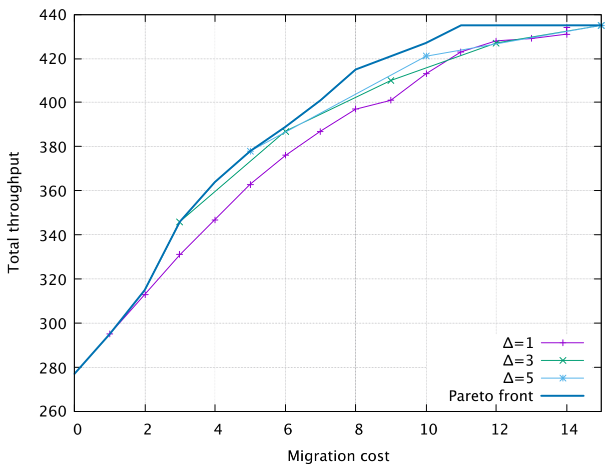

Fig. 5 Pareto front approximation: Exact Pareto front and approximate Pareto fronts from Algorithm 2 for $\Delta \in \{1, 3, 5\}$. Tree topology, 20 machines, 40 tasks.

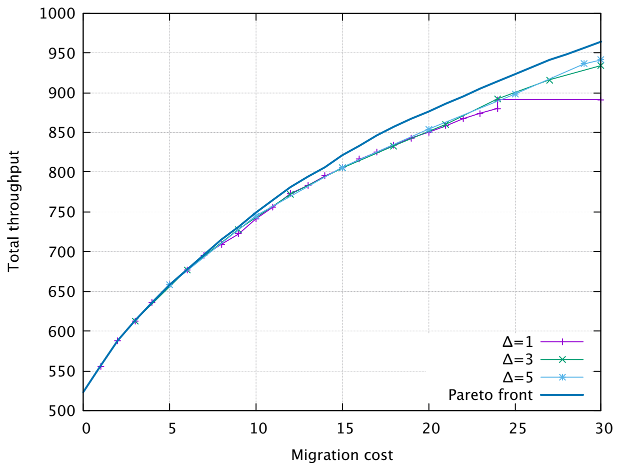

Fig. 6 Pareto front approximation: Exact Pareto front and approximate Pareto fronts from Algorithm 2 for $\Delta \in \{1, 3, 5\}$. Tree topology, 20 machines, 100 tasks.

follow the exact front. It can also be observed that for $\Delta = 1$, results are of lower quality, and that in the example of Figure 6, the algorithm was unable to find solutions for more than 24 migrations.

In order to quantify the impact of these algorithms on the quality of the obtained solution set, let θ_{20} (resp. θ_0, θ_{30}) represent the throughput corresponding to a migration cost of 20 (resp. 0, 30) in the Pareto front. More formally, this means that θ_{20} is the coordinate such that $(|I|, \theta_{20}, 20) \in \mathcal{S}$; for instance, in Figure 6, $\theta_{20} = 876$ for the exact Pareto front. Figure 7 (resp. 8) shows the *throughput improvement* θ_{20}/θ_0 (resp. θ_{30}/θ_0) for the 20 tested sce-

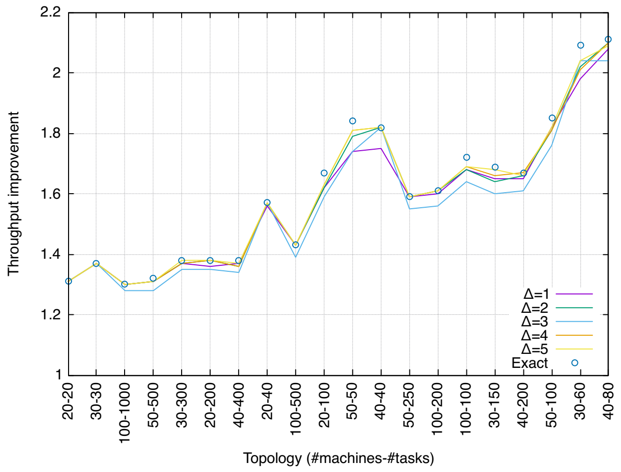

Fig. 7 Pareto front approximation: throughput improvement for 20 migrations θ_{20}/θ_0. Exact Pareto front vs approximate Pareto fronts from Algorithm 2 for $\Delta \in \{1, 2, 3, 4, 5\}$. 20 different scenarios.

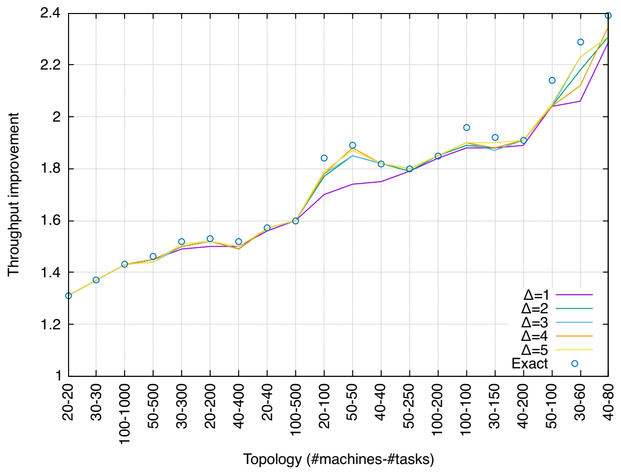

Fig. 8 Pareto front approximation: throughput improvement for 30 migrations θ_{30}/θ_0. Exact Pareto front vs approximate Pareto fronts from Algorithm 2 for $\Delta \in \{1, 2, 3, 4, 5\}$. 20 different scenarios.

narios, for the exact Pareto front as well as the approximate Pareto fronts. This represents the relative improvement in terms of overall throughput if the operator chooses to migrate 20 (resp. 30) tasks after inspecting the set of Pareto-optimal solutions. It appears that, for $\Delta \geq 2$, the achievable throughput improvement when using the heuristic method introduced in section 5.1 is close to the exact solution. Overall, choosing $\Delta = 5$ yields the better results: for 20 migrations, the results are better in all the twenty tested instances, and for 30 migrations they are outperformed by $\Delta = 4$ in only three instances. For

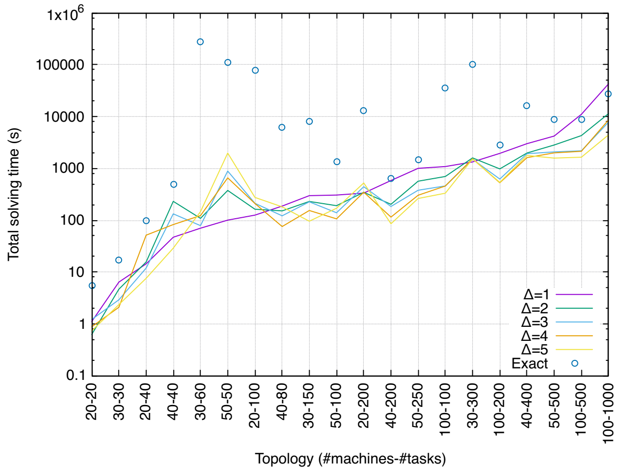

Fig. 9 Pareto front approximation: time to generate the approximate Pareto front, for up to $B = 30$ migrations. Exact Pareto front (Algorithm 1) vs approximate Pareto fronts (Algorithm 2) for $\varDelta \in \{1, 2, 3, 4, 5\}$. 20 different scenarios.

$\varDelta = 5$, the throughput improvement remains within a 3% error as compared to the exact solution after 20 migrations, and within a 5% error after 30 migrations. Using $\varDelta = 1$ yields results of lower quality, because the algorithm is not able to explore reconfigurations with two migrations at the same time (*i.e.* swapping tasks), and because it can terminate prematurely if the throughput cannot be improved by migrating only one task. For 20 migrations, $\varDelta = 3$ also yields results of lower quality, but this is more of an artificial nature: the number of migrations considered is 18 instead of 20 (since 20 is not a multiple of 3).

Finally, Figure 9 depicts the time needed to generate the exact and approximate Pareto fronts. While computing the exact Pareto front is very time-consuming, the approximation heuristics introduced reduces the required computation time. Overall, the lowest computation times are achieved for $\varDelta \in \{3, 4, 5\}$, whereas $\varDelta \in \{1, 2\}$ exhibit higher running times. The choice $\varDelta = 5$, which can be observed to exhibit the best results, in terms of throughput improvement, in the majority of cases – with a computation time constantly better than the reference exact approach, performing approximately one order of magnitude faster when $B = 30$ (from 5.1× faster for 100 machines and 200 tasks, to 1800× faster for 30 machines and 60 tasks).

To conclude, the approximation method presented in section 5.1 can greatly reduce the time needed to compute a set of approximate Pareto optimal solutions, while generating solutions close enough to the exact ones. It is realistic to note that, although an improvement in solving time is observed regardless of the topology size, Figure 9 suggests that using a MILP approach (exact or approximate) might be unfit for too large topologies.

6 Conclusion

In data centers, applications may exhibit time-varying communications dependencies, and migrating tasks in such environments can be a means to redistribute the load over the whole infrastructure, while optimizing inter-task communications. In this paper, a linear programming framework is introduced, which represents the data center topology, and communications dependencies between workloads. Using this framework, a multi-objective model is developed, which, when solved, can propose a reorganization of workloads so as to (i) place the highest number of new tasks, (ii) maximize the overall achievable throughput, and (iii) minimize the migration cost, incurred by this reorganization.

The approach developed is generic with respect to the network topology and the inter-application communication demands, and is novel in that, when compared to other workload migration frameworks, it generates a set of Pareto-optimal solutions. From among this solution set, a workload allocation can be chosen, which yields the best tradeoff in terms of throughput improvement versus the cost incurred by task migration. A solution algorithm using the ε-constraint method is proposed as a baseline. Then, a heuristic is presented, which generates an approximate Pareto front. Computational experiments on different topologies show that the proposed approximation builds solutions close to the optimal Pareto front, while greatly reducing the computation time.

Having shown the desirability of considering inter-application dependencies in migration models, an interesting question that arises is how the proposed model can be extended to take into account more fine-grained parameters. These could include the distance between source and destination hosts when migrating, the energy cost induced by machines which are on or off, and the possibility to load-balance flows across multiple paths.

References

1. Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286. USENIX Association, 2005.
2. Raffaele Bolla, Marco Chiappero, Riccardo Rapuzzi, and Matteo Repetto. Seamless and transparent migration for tcp sessions. In *Personal, Indoor, and Mobile Radio Communication (PIMRC), 2014 IEEE 25th Annual International Symposium on*, pages 1469–1473. IEEE, 2014.
3. Shripad Nadgowda, Sahil Suneja, Nilton Bila, and Canturk Isci. Voyager: Complete container state migration. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 2137–2142. IEEE, 2017.
4. Dazhao Cheng, Changjun Jiang, and Xiaobo Zhou. Heterogeneity-aware workload placement and migration in distributed sustainable datacenters. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 307–316. IEEE, 2014.
5. Deze Zeng, Lin Gu, and Song Guo. Cost minimization for big data processing in geo-distributed data centers. In *Cloud Networking for Big Data*, pages 59–78. Springer, 2015.

6. Vivek Shrivastava, Petros Zerfos, Kang-Won Lee, Hani Jamjoom, Yew-Huey Liu, and Suman Banerjee. Application-aware virtual machine migration in data centers. In *INFOCOM, 2011 Proceedings IEEE*, pages 66–70. IEEE, 2011.
7. Daochao Huang, Yangyang Gao, Fei Song, Dong Yang, and Hongke Zhang. Multi-objective virtual machine migration in virtualized data center environments. In *Communications (ICC), 2013 IEEE International Conference on*, pages 3699–3704. IEEE, 2013.
8. Theophilus Benson, Aditya Akella, and David A Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 267–280. ACM, 2010.
9. Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 202–208. ACM, 2009.
10. Xiaoqiao Meng, Vasileios Pappas, and Li Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.
11. Katrina LaCurts, Shuo Deng, Ameesh Goyal, and Hari Balakrishnan. Choreo: Network-aware task placement for cloud applications. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 191–204. ACM, 2013.
12. Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *NSDI*, volume 10, pages 19–19, 2010.
13. Tiago C. Ferreto, Marco A.S. Netto, Rodrigo N. Calheiros, and César A.F. De Rose. Server consolidation with migration control for virtualized data centers. *Future Generation Computer Systems*, 27:1027–1034, 2011.
14. Chaima Ghribi, Makhlouf Hadji, and Djamal 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.
15. Hao Jin, Tosmate Cheocherngngarn, David Levy, Alex Smith, Deng Pan, Jiangchuan Liu, and Niki Pissinou. Joint host-network optimization for energy-efficient data center networking. In *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 623–634. IEEE, 2013.
16. Ning Liu, Ziqian Dong, and Roberto Rojas-Cessa. Task and server assignment for reduction of energy consumption in datacenters. In *Network Computing and Applications (NCA), 2012 11th IEEE International Symposium on*, pages 171–174. IEEE, 2012.
17. Dhaemesh kakadia, Nandish Kopri, and Vasudeva Varma. Network-aware virtual machine consolidation for large data centers. In *NDM '13 Proceedings of the Third International Workshop on Network-Aware Data Management.* ACM (6), 2013.
18. Raja Wasim Ahmad, Abdullah Gani, Siti Hafizah Ab. Hamide, Muhammada Shiraz, Abdullah Yousafzai, and Feng Xia. A survey on virtual machine migration and server consolidation frameworks for cloud data centers. *Journal of Network and Computer Applications*, 52:11–25, 2015.
19. Fabio López Pires and Benjamin Báran. Virtual machine placement literature review. arXiv:1506.01509v1, cited 4 June 2015.
20. Md Hasanul Ferdaus, Manzur Murshed, Rodrigo N Calheiros, and Rajkumar Buyya. Network-aware virtual machine placement and migration in cloud data centers. *Emerging Research in Cloud Distributed Computing Systems*, 42, 2015.
21. Zoha Usmani and Shailendra Singh. A survey of virtual machine placement techniques in cloud data center. *Procedia Computer Science*, 78:491–498, 2016.
22. Weiwei Fang, Xiangmin Liang, Shengxin Li, Chiaraviglio, and Naixue Xiong. VMPlanner: Optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers. *Computer Networks*, 57(1):179–196, 2013.
23. Tao Chen, Xiaofeng Gao, and Guihai Chen. Optized virtual machine placement with traffic-aware balancing in data ceter networks. *Scientific Programming*, 6:10 pages, 2016.

24. Shuo Fang, Renuga Kanagavelu, Bu-Sung Lee, Chuan Heng Foh, and Khin Mi Mi Aung. Power-efficient virtual machine placement and migration in data centers. In *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, pages 1408–1413. IEEE Computer Society, 2013.
25. Jing Xu and José A.B. Fortes. Multi-objective virtual machine placement in virtualized data center environments. In *2010 IEEE/ACM International Conference on Green Computing and Communications & 2010 IEEE/ACM International Conference on Cyber, Physical and Social Computing*, pages 179–188. IEEE Computer Society, 2010.
26. Jing Xu and José A.B. Fortes. A multi-objective approach to virtual machine management in datacenters. In *Proceeding ICAC'11 Proceedings of the 8th ACM international conference on Autonomic computing*, pages 225–234. ACM New York, NY, USA, 2011.
27. Yongquiang Gao, Haibing Guan, Zhengwei Qi, Yang Hou, and Liang Liu. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *Journal of Computer and System Sciences*, 79:1230–1242, 2013.
28. Fabio López Pires and Benjamin Báran. Multi-objective virtual machine placement with service level agreement. In *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 203–210. IEEE Computer Society, 2013.
29. Fabio López Pires and Benjamin Báran. Virtual machine placement. A multi-objective approach. In *Latin American Symposium of Infrastructure, Hadward, and Software*, pages 77–84. IEEE, 2013.
30. Matthias Ehrgott. *Multicriteria optimization*. Springer Science & Business Media, 2006.
31. Ravindra K Ahuja, Thomas L Magnanti, James B Orlin, and MR Reddy. Applications of network optimization. *Handbooks in Operations Research and Management Science*, 7:1–83, 1995.
32. Marco Laumanns, Lothar Thiele, and Eckart Zitzler. An adaptive scheme to generate the pareto front based on the epsilon-constraint method. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2005.
33. Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2015.