Partition with side effects

Fanny Pascual
Sorbonne Universités
UPMC (Université Paris 6)
LIP6, CNRS, UMR 7606
Email: fanny.pascual@lip6.fr

Krzysztof Rzadca
Institute of Informatics
University of Warsaw
Warsaw, Poland
Email: krz@mimuw.edu.pl

***Abstract*—In data centers, many tasks (services, virtual machines or computational jobs) share a single physical machine. We propose a new resource management model for such colocation. Our model uses two parameters of a task—its size and its type—to characterize how a task influences the performance of the other tasks allocated on the same machine. As typically a data center hosts many similar, recurring tasks (e.g.: a webserver, a database, a CPU-intensive computation), the resource manager should be able to construct these types and their performance interactions. Moreover, realistic variants of our model are polynomially-solvable, in contrast to the NP-hard vector packing used previously. In particular, we minimize the total cost in a model in which each task's cost is a function of the total sizes of tasks allocated on the same machine (each type is counted separately). We show that for a linear cost function the problem is strongly NP-hard, but polynomially-solvable in some particular cases. We propose an algorithm polynomial in the number of tasks (but exponential in the number of types and machines); and another algorithm polynomial in the number of tasks and machines (but exponential in the number of types and admissible sizes of tasks). When there is a single type, we give a polynomial time algorithm. We also prove that, even for a single type, the problem becomes NP-hard for convex costs.**

***Keywords*-data center; heterogeneity; resource management; scheduling; colocation; co-tenancy; partition; complexity; algorithm**

I. Introduction

Data centers, composed of tens to hundreds of thousands of machines, packaged as virtual machines or services and sold under the label of cloud, are now changing the way the industry (and, to some extent, academia and research) computes. Virtualization packages individual resources into standard chunks with performance guaranteed by Service Level Agreements (SLAs). Economies of scale make the whole endeavour profitable for huge companies, like Google, or providers of for-hire computational power (such as Amazon EC2, RackSpace or Google Compute Engine).

However, a data center is not just a standard HPC supercomputer, but used by Google instead of a university. In their great majority, HPC workloads are composed of computationally-intensive batch jobs (recent workloads may be also memory-intensive, which requires significant changes to HPC resource managers [1]). The goal of an HPC scheduler is to order jobs so that they are completed as fast as possible, taking into account site's policies, fairness and efficiency. As jobs are computationally-intensive, they all compete for the same resource—the CPU—so, a cluster node executes at most as many jobs as CPU cores.

In contrast, a data center workload is more varied. In the Google trace, just 1.5% of applications contribute 98.5% of CPU usage [2]. Thus, while there are some computationally-intensive batch jobs (corresponding to, e.g., Google's Page rank recalculation), a large part of the workload are services (sometimes run in a virtualized environment). A service models, e.g., a single instance of a web application serving users (network-intensive); or an instance of a database (IO- or memory-intensive). (In order not to distinguish at a semantic level between a service, a virtual machine that may run it, and a computational job, we will later refer to them as *tasks*).

These new features of data center workloads make HPC models unsuitable for managing data center's resources. As tasks require heterogeneous resources [3] (CPU, memory, hard disk bandwidth, network bandwidth), sharing a single node among many services is reasonable. Ideally, colocated tasks' resource requirements should complement each other, e.g., a memory-intensive database instance should be allocated with a few IO-intensive web applications with burst popularity. The goal of the resource manager is also different: instead of completing tasks as fast as possible, the resource manager should optimize the end-user experience (measured as, e.g., a statistics of the response time, e.g., time by which 95 or 99 percent of requests are completed). However, the responsme has a non-linear dependency on the load of the node [4], [5], [6].

Our model. In this paper, we propose a new model that captures both complex goal functions and the effects that tasks have on each other when they are allocated to the same machine. In classic scheduling, a task's influence on other tasks depends solely on its size (which represents its load, or its processing time). We propose a notion of *type* of a task generalizing its side-effects. A task influences other tasks in function of its size *and its type*. Thus, the cost of a task i assigned to a machine M is a function of the size of the tasks of each type on that machine (if there are

T possible types in the system, this cost function takes T arguments, the x^{th} argument being the total size of the tasks of type x on M). In this paper (except in Section IV-B), we consider a linear cost function. Linear costs roughly correspond to classic optimality measures; and non-linear costs model complex end-user performance. In linear cost function, task i's cost is the sum of the *weighted* sizes of all tasks j assigned to machine M. The weight, which depends on i's and j's types, measures the compatibility between the two types. Large weights correspond to types that compete for similar resources. Small weights correspond to types that complement each other, e.g., a CPU-intensive and a memory-intensive tasks.

Our model has three main advantages: first, it captures tasks' heterogeneity; second, it optimizes the observed (experienced) performance of the tasks, and not just the usage of the resources; third, it is a minimal extension of a standard scheduling model. Tasks' affinities or interferences, similar to our notion of type, were proposed in recent papers on colocation performance [7], [8] (see Section V). In contrast to bin-packing models, we don't use a strict limit on machines' capacities. Hardware resources are limited, but a task does not abruptly fail when, e.g., the total cpu usage (or the disk IO, bandwidth or even memory, with OS swapping) gets to 100%. Instead, tasks' performance is gradually degraded, resulting in slower observed response times. Additionally, some tasks (e.g. webservers [4]) have better observed performance if the total cpu usage is 20-40%, rather than 90-100%. The aim of our model is to go beyond the crisp constraints of bin-packing, which unrealistically treats any packing not exceeding the capacity as equally good, while not permitting even small overpacking.

Contribution. The main contribution of our paper is a mathematical model that captures the effects that heterogeneous tasks have on each other when they are allocated to the same machine. The detailed contributions are as follows:

- We propose the problem of Partition with Side Effects (PSE), in which the cost of a task is a linear function of the size of the other tasks on the same machine (sizes are multiplied by factors representing the compatibility between types).
- We prove that PSE is NP-hard for arbitrary number of types (Section III-A), and this even for two machines and tasks of the same size. As a type corresponds to the type of influence of a group of tasks on other groups, a consequence of this result is that the number of influences (types) considered should be small (constant or logarithmic with the number of tasks).
- We show that there is an optimal solution in which, for each type t_i, there is an order O_i of the machines in which the tasks sorted in order of non-increasing sizes are assigned to the machines sorted in order O_i (Section III-C).
- Using the above dominance rule, we derive an exact algorithm polynomial in the number of tasks (but exponential in the number of types and machines, Section III-D).
- We also give a dynamic programming algorithm polynomial in the number of tasks and machines (but exponential in the number types and of sizes of tasks, Section III-B).
- For a single type, we give an exact algorithm polynomial in the number of tasks and machines (Section IV-A).
- We show that if the costs are convex (and not linear, as used above), the problem becomes NP-hard even for a single type (Section IV-B).

The paper is organized as follows. We define the problem of partition with side effects in Section II. We then study two variations of this problem. Section III analyzes the case with tasks of multiple types and costs linear with the sizes of tasks. Section IV analyzes the case with tasks of single type and costs that are linear or convex. Section V discusses the related work.

II. Model

We consider a system where n tasks $J = \{1, \ldots, n\}$ have to be allocated on a set of m parallel identical machines $\mathcal{M} = \{M_1, \ldots, M_m\}$. Each task i has a known size $p_i \in \mathbb{N}$ (this assumption corresponds to the widely used clairvoyant scheduling model: the sizes can be estimated by the resource manager using previous instances, or users' estimates). The size corresponds to load the task imposes on a machine: the request rate for a web server; or the cpu load for a cpu-intensive computation. We assume that the tasks are indexed by non-increasing sizes: $p_1 \geq p_2 \geq \cdots \geq p_n$.

A *partition* (also called an *allocation*) is an assignment of each of the n tasks to one of the m machines. In other words, a partition divides the tasks into at most m subsets, each subset corresponding to the tasks allocated on the same machine. Given a partition P, we denote by $M_{P,i} \in \mathcal{M}$ the machine on which task i is allocated in P.

The *load* of a machine M_j in a partition P is the sum of the sizes of the tasks assigned to M_j in P. The *load of type t* of a machine M_j in a partition P is the sum of the sizes of the tasks of type t assigned to M_j in P. The cost c_i of task i in P depends on the loads of different types of $M_{P,i}$, the machine task i is allocated to (we later formalize c_i). The cost of partition P, denoted by $C(P)$, is the sum of the costs of the tasks: $C(P) = \sum_{i=1}^{n} c_i$. Our aim is to compute a partition of minimum cost. Such a partition minimizes the average cost of a task and thus corresponds to the socially-optimal outcome in the utilitarian model.

The main contribution of this paper lies in modeling side-effects of colocating tasks on a single machine. The impact of task i on the cost of another task j is a function of task's size p_i and *task's type* t_i. Types generalize tasks' impact on the performance and may have different granularities: for instance, "a webserver" and "a database"; or "a read-intensive MySQL database"; or, as in [7], "an instance of Blast" . Let $\mathcal{T} = \{1, \ldots, T\}$ be a set of T different types

of tasks. Each task i has type $t_i \in \mathcal{T}$. Here we again assume the clairvoyant model: task's type is known to the resource manager either by analysis of previous instances, or by users' declarations. For each type $t \in \mathcal{T}$, we denote by J^t the tasks which are of type t; by $n^{(t)}$ the number of such tasks ($n^{(t)} = |J^t|$); by j_i^t the i-th largest task of type t (ties are broken arbitrarily); and by p_i^t this task's size.

Different types have different influence on the cost of a task. In most of the paper (except Section IV-B) we use a linear cost function:

$$c_i = \sum_{j \text{ on machine } M_{P,i}} p_j.\alpha_{t_j,t_i},$$

where a coefficient $\alpha_{t,t'} \in \mathbb{N}$, defined for each pair of types $(t,t') \in \mathcal{T}^2$, measures the impact of the tasks of type t on the cost of the tasks of type t' (allocated on the same machine). If $\alpha_{t,t'} = 0$ then a task of type t has no impact on the cost of a task of type t'; the higher the $\alpha_{t,t'}$, the larger the impact. Coefficients are not symmetric, i.e., it is possible that $\alpha_{t,t'} \neq \alpha_{t',t}$. We consider the linear cost function as it generalizes, by adding coefficients $\alpha_{t,t'}$, one of well-studied scheduling models [9], [10], in which the cost of a task is the load of its machine (i.e., if $\forall (t,t') \in \mathcal{T}^2 \alpha_{t,t'} = 1$, the model reduces to the classical model). The coefficients $\alpha_{t,t'}$ can be estimated by monitoring tasks' performance in function of their colocation and their sizes, which should be feasible as a data center runs many instances of similar services [7], [8].

We denote by PSE (Partition with Side Effects) the problem of minimizing the total cost $C(P)$ of partition P, $C(P) = \sum_{i=1}^{n} c_i$, with c_i defined by the linear cost function.

In Section IV-B, we study an important generalization of the cost model, in which the cost of a task is any convex function of the total load of the machine.

III. Several types

In this section we analyze the Partition with Side Effects (PSE) problem in the general case (although always using the linear cost function; we relax this assumption in Section IV-B).

We start by analyzing the complexity of the problem: Section III-A shows that PSE is NP-hard for arbitrary number of types even for two machines and unit-size tasks. Section III-B proposes a dynamic programming algorithm polynomial in the number of tasks and machines, but exponential in the number of admissible sizes of tasks and in the number of types. Then, in Section III-C, we give a dominance rule: we show that there is an optimal solution in which, for each type t_i, there is an order O_i of the machines in which the tasks sorted in order of non-increasing sizes are assigned greedily to the machines sorted in order O_i. This property will be called *the SPT property* (where SPT stands for "Shortest Processing Time"). It allows us to construct another dynamic programming algorithm called CutJuxtapose (Section III-D), polynomial in the number of tasks, but exponential in the number of types and machines.

Before we analyze the complexity of PSE, we mention several special cases that can be efficiently solved using the results presented in Section IV.

One type / equivalent types: when there is only one type, or, alternatively, if there is a value C such that for each pair of types t and t' (including $t = t'$), $\alpha_{t,t'} = C$, we show in Section IV-A that PSE can be solved in $O(n^2 m)$.

Independent types: when for all $i \neq j$ $\alpha_{i,j} = 0$, the types are independent: by using T times the algorithm of Section IV-A (for each type we use this algorithm to assign optimally the tasks of the considered type to the m machines), we obtain an optimal solution in $O(n^2 m)$.

Large influences: If influences are very large for each pair of types $t' \neq t$ (i.e., $\forall (t',t) \in \mathcal{T}^2, t \neq t' : \alpha_{t',t} > \sum_{t \in \mathcal{T}} (n^{(t)} \alpha_{t,t} \sum_{i \in J^t} p_i)$), and if $T \leq m$, then sharing machines between tasks of different types is inefficient. Indeed, the cost of allocating two tasks of different types on a same machine incurs a cost larger than the total cost of a partition where the tasks do not share machines ($\sum_{t \in \mathcal{T}} n^{(t)} \alpha_{t,t} \sum_{i \in J^t} p_i$ is the cost of a partition using T machines, each dedicated to a specific type).

For a given possible configuration (assignment of a number of machines to each type, such that the total number of machines is m), we can compute an optimal partition in polynomial time, again by using T times the $O(n^2 m)$-algorithm of Section IV-A. We consider all the possible configurations, and the partition of minimum cost over all the configurations is returned. In order to count the number of possible configurations, we show how to generate all these configurations: since there is at least one machine per type, w.l.o.g. we assign machine $m - i + 2$ to type i, for each $i \in \{2, \ldots, T\}$ (at least one machine will be assigned to type 1 later). There remains $m - T + 1$ machines to assign. We throw $T - 1$ balls on the $m - T + 1$ first machines: let $k_1, \ldots, k_{T-1}$ be the $T - 1$ machines (sorted by non decreasing index) on which there is a ball (there may be several balls on the same machine). Machines with ID smaller than or equal to k_1 (there is at least such a machine) are assigned to the tasks of type 1; for each $j \in \{2, \ldots, T-1\}$, machines with ID larger than k_{j-1} and smaller than or equal to k_j are for the tasks of type j; machines with ID larger than k_{T-1} and at most $m - T + 1$ are for the tasks of type T. The number of configurations is equal to the number of way to throw $T-1$ balls on $m-T+1$ machines, i.e., the number of $(T-1)$-combinations with repetitions from a set of size $m - T + 1$, that is $\binom{(m-T+1)+(T-1)-1}{(T-1)} = \binom{m-1}{T-1}$. Thus the complexity of our algorithm is $O(\binom{m-1}{T-1} n^2 m) \subset O(n^2 m^T)$. This is a polynomial time algorithm when T is a constant.

A. Complexity with arbitrary number of types

Proposition III-A.1: The decision version of PSE is strongly NP-complete if the number of types is not fixed, and this even if there are only two machines and if all the tasks have unit size.

Proof: We reduce strongly NP-complete SIMPLE MAX CUT [11] to PSE. The SIMPLE MAX CUT problem is the following one: given a graph $G = (V, E)$ and a positive integer K, is there a partition of V into two disjoints sets V_1 and V_2 such that the number of edges that have one endpoint in V_1 and the other endpoint in V_2 is at least K ?

The decision version of PSE is as follows: given an instance of PSE, and a bound B, is there a partition with cost at most B ? We construct an instance of PSE from an instance of SIMPLE MAX CUT as follows: we have two machines M_1 and M_2, and $n = |V|$ tasks $\{1, \ldots, n\}$, each one of size 1, and of a different type. We label the vertices of V by the integers from 1 to $|V|$. Each task i corresponds to vertex i of V. The values α correspond to edges: for each $(i, j) \in V^2$, $\alpha_{t_i,t_j} = \frac{1}{2}$ if $\{i, j\} \in E$ and $\alpha_{t_i,t_j} = 0$ if $\{i, j\} \notin E$. We fix $B = |E| - K$.

We first show that there is a solution to PSE if there is a solution to SIMPLE MAX CUT. Assume that there is a solution to the SIMPLE MAX CUT: let V_1 and V_2 be two sets such that the number of edges that have one endpoint in V_1 and one endpoint in V_2 is at least K. We construct a partition P for PSE by assigning the tasks corresponding to vertices in V_1 (resp. V_2) to machine M_1 (resp. M_2). For each task $i \in \{1, \ldots, n\}$, let $n_i = \sum_{j \in M_{P,i} | \{i,j\} \in E} 1$ (for each task i on M_1 (resp. on M_2), n_i is the number of neighbours of vertex i in V_1 (resp. in V_2)). The cost of task i is $c_i = \sum_{j \in M_{P,i}} \alpha_{t_j,t_i} p_j = \frac{1}{2} \sum_{j \in M_{P,i} | \{i,j\} \in E} 1 = \frac{1}{2} n_i$. The total cost of the allocation P is $C(P) = \sum_{i=1}^{n} c_i = \frac{1}{2} \sum_{i=1}^{n} n_i$. Note that $\sum_{i=1}^{n} n_i$ is twice the number of edges for which both endpoints are in the same set (an edge $\{i, j\}$ between two vertices in the same set adds 1 to n_i and 1 to n_j). Therefore $\frac{1}{2} \sum_{i=1}^{n} n_i$ is at most $|E| - K$ since there are at least K edges which have an endpoint in V_1 and an endpoint in V_2. Thus, $C(P) = \frac{1}{2} \sum_{i=1}^{n} n_i \leq |E| - K = B$. There is a solution to PSE.

Assume now that there is a partition P of cost at most B for PSE. We partition the set $V = \{1, \ldots, n\}$ into two sets V_1 and V_2 following the partition P: if task i is assigned to machine M_1 (resp. M_2) in P then vertex i belongs to V_1 (resp. i belongs to V_2). The total cost $C(P)$ is equal to the sum of the costs of the tasks. The cost of a task i is $\sum_{j \in M_{P,i}} \alpha_{t_j,t_i} p_j = \frac{1}{2} \sum_{j \in M_{P,i} | \{i,j\} \in E} 1$. Therefore, the cost of P is equal to the number of edges which have both endpoints either in V_1 or in V_2. Since the cost of P is at most $B = |E| - K$, then there are at least K edges which have an endpoint in V_1 and an endpoint in V_2: there is a solution to SIMPLE MAX CUT.

There is a solution to PSE if and only if there is a solution to SIMPLE MAX CUT. As PSE is (trivally) in NP, and as SIMPLE MAX CUT is strongly NP-complete, PSE is also strongly NP-complete. ■

B. Allocation for a fixed number of sizes

We now present a dynamic programming algorithm which solves PSE in polynomial time if the number of types is constant and if the number of possible sizes for the tasks is also constant.

For each type $t \in \mathcal{T}$, we denote by l_t the number of different sizes of a task of type t. Each possible size of a task of type t has a index in $\{1, \ldots, l_t\}$. The size of a task of type t and of size index $j \in \{1, \ldots, l_t\}$ is p_t^j (note that in this subsection we deliberately put the type t in subscript in order not to confuse p_t^j, the size of the task with the size index j, with p_j^t, the size of the j-th largest task of type t). We denote by $\mathcal{C}_t^j$ the set of the tasks which are of type t and of size p_t^j. The number of such tasks is $n_t^j = |\mathcal{C}_t^j|$.

By $C(y_1^1, \ldots, y_1^{l_1}, y_2^1, \ldots, y_2^{l_2}, \ldots, y_T^1, \ldots, y_T^{l_T}, r)$ we denote the cost of an optimal solution of problem PSE when there are r machines and y_t^j tasks of type t and of size p_t^j. For short, we will denote this cost by $C\left((y_t^j), r\right)$ (with $t \in \{1, \ldots, T\}$ and $j \in \{1, \ldots, l_t\}$). The following dynamic programming algorithm (proved in Proposition III-B.1) finds in polynomial time an optimal solution of PSE.

The cost of an optimal allocation on a single machine is:

$$C\left((y_t^j), 1\right) = \sum_{t=1}^{T} \sum_{t'=1}^{T} \alpha_{t,t'} \left(\sum_{k=1}^{l_t} y_t^k p_t^k \right) \left(\sum_{k=1}^{l_{t'}} y_{t'}^k \right). \quad (1)$$

The cost of an optimal allocation on $r \geq 2$ machines is:

$$C\left((y_t^j), r\right) = \min_{(x_t^j : x_t^j \in \{0, \ldots, y_t^j\})} \left(C\left((y_t^j - x_t^j), r - 1\right) + C\left((x_t^j), 1\right) \right). \quad (2)$$

The cost of an optimal solution of PSE is $C((n_t^j), m)$. As usual, by backtracking we can deduce from $C((n_t^j), m)$ an allocation of minimal cost.

Proposition III-B.1: If the number of types and the number of possible sizes for the tasks are constant, the above described dynamic programming algorithm optimally solves PSE in $O(n^{2 \sum_{t \in \mathcal{T}} l_t} m)$.

Proof: When there is a single machine, there is only one possible allocation, whose cost is the sum, over all the couples of types $(t, t') \in \mathcal{T}^2$, of the cost that the tasks of type t imply on the cost of the tasks of type t' (i.e., the load of the tasks of type t times the number of tasks of type t', times $\alpha_{t,t'}$). This cost is expressed by the right hand side of Equation 1. The value $C\left((y_t^j), 1\right)$ is thus valid.

For each $t \in \{1, \ldots, T\}$, and then for each $j \in \{1, \ldots, l_t\}$, let $y_t^j \in \{0, \ldots, n_t^j\}$ be a number of tasks

of type t and of size p_t^j, and let $x_t^j \in \{0, \ldots, y_t^j\}$. The expression $C\left((y_t^j - x_t^j), r-1\right) + C\left((x_t^j), 1\right)$ computes the cost of an optimal solution among the solutions where there are x_t^j tasks of type t and of size p_t^j on machine M_r, and where there are $y_t^j - x_t^j$ tasks of type t and of size p_t^j on machines M_1 to M_{r-1}. In any partition of the tasks (y_t^j), the number of tasks of type t and of size p_t^j on M_r is between 0 and y_t^j. Thus, the right hand side of Equation 2 computes the optimal cost of a partition where there are y_t^j tasks of type t and of size p_t^j to assign to r machines, and Equation 2 is valid.

Therefore, by using Equations 1 and 2, we can compute $C((n_t^j), m)$, the cost of an optimal solution of problem PSE. Let us now analyze the time complexity of this algorithm.

Since, on a given subset of the machines, the number of tasks of type t and size p_t^j is between 0 and n_t^j, the number of possible vectors (y_t^j) is $\prod_{t\in\mathcal{T}, j\in\{1,\ldots,l_t\}}(n_t^j + 1) < (n+1)^{\sum_{t\in\mathcal{T}} l_t}$. Thus, the number of possible values of $C((y_t^j), r)$ that we have to compute is smaller than $m(n+1)^{\sum_{t\in\mathcal{T}} l_t}$. Each value $C((y_t^j), 1)$ is computed in $O(T^2(\max_t l_t)^2)$, which is a constant time since T and l_t are constant. Each value $C((y_t^j), r)$, with $r \geq 2$, is computed at most in $(n+1)^{\sum_{t\in\mathcal{T}} l_t}$ (once the values C with less than r machines have been computed and stocked). Therefore, if the number of types and the number of possible sizes for the tasks are constant, the complexity of this dynamic programming algorithm is $O(n^{2\sum_{t\in\mathcal{T}} l_t} m)$: it is a polynomial time algorithm. ■

C. The Shortest Processing Time (SPT) Property

An allocation fulfills the *shortest processing time (SPT) property* if, for each type t_i, there is an order O_i of the machines in which the tasks sorted in order of non-increasing sizes are assigned to the machines sorted in order O_i. As we demonstrate here, there is an optimal allocation which fulfills the SPT property. This dominance rule will allow us to derive an optimal algorithm in the next section.

More formally, for each type t_i, if two tasks s (short) and l (long) of type t_i are assigned to machine M_i, then the other machines are not assigned tasks of type t_i of intermediate sizes (sizes larger than p_s but smaller than p_l). In other words, an allocation fulfills the SPT property if, for each type q and each machine $M_i \in \mathcal{M}$ the tasks of type q assigned to M_i are a single, contiguous sublist of a list $(i : t_i = q)$ where the tasks are sorted by non-increasing sizes. We say that a triple of tasks (s, l, x) such that $p_s < p_x < p_l$ breaks the SPT property if tasks s and l are allocated to the same machine whereas task x is allocated to another machine.

Lemma III-C.1: For each instance of PSE there exists an optimal allocation which fulfills the SPT property.

Proof: Let P be an optimal allocation for a given instance I of PSE. If P fulfills the SPT property then the proof is complete. Otherwise, let us transform P into an allocation of the same cost and which fulfills the SPT property. As P does not fulfill the SPT property, there are three tasks s (small) x (medium) and l (large) of type $t \in \mathcal{T}$ such that $p_s < p_x < p_l$ and such that tasks s and l are in P on the same machine, M_i, and task x is on another machine $M_j \neq M_i$.

Let us denote by P_{s-x} an allocation in which s and x are exchanged (i.e., task s is on M_j, task x is on M_i, and the remaining tasks $k \notin \{s, x\}$ are on the same machines as in P, $M_{P_{s-x},k} = M_{P,k}$). Likewise, we denote by P_{l-x} an allocation in which l and x are exchanged. We now show that the costs of the partitions P_{s-x} and P_{l-x} are equal to the cost of P.

For each type $q \in \mathcal{T}$, and for each machine $M \in \mathcal{M}$, the number of tasks of type q on M is the same in P, P_{s-x}, and P_{l-x}. For each type $q \in \mathcal{T}$, let us denote by n_i^q (resp. n_j^q) the number of tasks of type q on M_i (resp. M_j).

For each task a which is not allocated to M_i or M_j in P, the cost of a is the same in P, P_{s-x}, and P_{l-x}. Indeed, the cost of a task depends only on the tasks allocated on the same machine, and the partitions P, P_{s-x}, and P_{l-x} are identical on all the machines except machines M_i and M_j. In P_{s-x} on M_i, the loads of the other types remain the same, so the cost of each task of type q increases exactly by $(p_x - p_s)\alpha_{t,q}$ compared to its cost in P, since the exchange of s and x *increases* on M_i the load of type t by $(p_x - p_s)$. Likewise, in P_{s-x} on M_j, the cost of each task of type q decreases by $(p_x - p_s)\alpha_{t,q}$ compared to its cost in P, since the exchange of s and x *decreases* on M_j the load of type t by $(p_x - p_s)$. Therefore, we have:

$$\begin{aligned} cost(P_{s-x}) &= cost(P) + \sum_{q\in\mathcal{T}} ((p_x - p_s).\alpha_{t,q}.n_i^q) + \\ &\quad + \sum_{q\in\mathcal{T}} \left((p_s - p_x).\alpha_{t,q}.n_j^q\right) \\ &= cost(P) + (p_x - p_s) \sum_{q\in\mathcal{T}} \left(\alpha_{t,q}.(n_i^q - n_j^q)\right) \end{aligned}$$

Likewise,

$$\begin{aligned} cost(P_{l-x}) &= cost(P) + \sum_{q\in\mathcal{T}} ((p_x - p_l).\alpha_{t,q}.n_i^q) + \\ &\quad + \sum_{q\in\mathcal{T}} \left((p_l - p_x).\alpha_{t,q}.n_j^q\right) \\ &= cost(P) + (p_x - p_l) \sum_{q\in\mathcal{T}} \left(\alpha_{t,q}.(n_i^q - n_j^q)\right) \end{aligned}$$

If $\sum_{q\in\mathcal{T}} \left(\alpha_{t,q}.(n_i^q - n_j^q)\right) < 0$ then $cost(P_{s-x}) < cost(P)$, which is impossible since P is optimal. Likewise, if $\sum_{q\in\mathcal{T}} \left(\alpha_{t,q}.(n_i^q - n_j^q)\right) > 0$ then $cost(P_{l-x}) < cost(P)$, which again is impossible since P is optimal. Thus,

$$\sum_{q\in\mathcal{T}} \left(\alpha_{t,q}.(n_i^q - n_j^q)\right) = 0,$$

and $cost(P) = cost(P_{l-x}) = cost(P_{s-x})$.

We now consider an optimal allocation P_{s-x}. If P_{s-x} fulfills the SPT property, we are done. Otherwise we repeat the above algorithm (starting with P_{s-x} as the current allocation) until the obtained partition fulfills the SPT property. As, at each step we decrease the number of triples of tasks breaking the SPT property, the algorithm converges. Since the cost of the allocation does not change, and since the original allocation P is optimal, we will obtain an optimal allocation which fulfills the SPT property. ■

D. CutJuxtapose: Partition using the SPT property

We show in this section an optimal algorithm for PSE, called CUTJUXTAPOSE (Algorithm 1) that uses the SPT property (Lemma III-C.1). CUTJUXTAPOSE is exponential in the number of types T and in the number of machines m, but polynomial in the number of tasks n.

As the SPT property specifies, for each type, an optimal ordering of tasks (from the largest to the smallest), CUTJUXTAPOSE "cuts", for each type t, the ordered sequence of tasks of type t into at most m sub-sequences (each subsequence corresponds to a set of tasks of type t assigned to the same machine). Then, CUTJUXTAPOSE "juxtaposes" (combines) sub-sequences belonging to different types (i.e., sub-sequences of a given type are allocated to different machines).

The main function, `CutJuxtapose`, cuts the sequence of tasks of type t, $(j_1^t, j_2^t, \ldots, j_{n^{(t)}}^t)$, into m sub-sequences using indices denoting the cut points $k_1^t, k_2^2, \ldots, k_{m-1}^t$ (these indices are sorted in non-decreasing order). There will be $m-1$ cut points among $n^{(t)}$ possible indices. All the possible subsequences should be generated, so the cut points are generated as combinations (to test all the possibilities) with replacement (to allow empty allocations for some machines: if some cut points are equal, the resulting sub-sequences are empty, corresponding to a type that do not use all the machines). The first sub-sequence spans tasks of type t with indices $(1, \ldots, k_1^t)$; the i-th subsequence (with $i < m$) spans the tasks of type t with indices $(k_{i-1}^t + 1, k_i^t)$ — this subsequence is empty if $k_{i-1}^t + 1 > k_i^t$; the last subsequence spans the tasks of type t with indices $(k_{m-1}^t + 1, n^{(t)})$. Given m sub-sequences for each type, the algorithm finds a optimal way to combine them by testing all sub-sequences' permutations (function `Juxtapose`). The algorithm allocates on the i-th machine the i-th sub-sequence of first type, J_i^1, combined with the i-th (after permutation) sub-sequence of the second type, $J_{\pi^2(i)}^2$, etc.

Proposition III-D.1: Algorithm CUTJUXTAPOSE computes in $O(n^{(m-1)T}(m!)^{T-1})$ an optimal allocation of problem PSE.

Proof: By Lemma III-C.1 we know that there exists an optimal partition which fulfills the SPT property. Algorithm CUTJUXTAPOSE consider all the possible SPT partitions: function `CutJuxtapose` consider all the possible ways to divide the tasks of each type into at most m subsequences, and then for each of these ways it computes the best partition where the tasks of each subsequence share the same machine. Indeed, given m sets of tasks for each type, function `Juxtapose` consider all the possible assignments of the sets to the machine (given that two sets of the same type are on different machines), and it returns the best partition. When the subsequences correspond to the ones of an optimal solution, the algorithm computes then an optimal solution PSE. This is thus the solution which is returned by the algorithm.

The number of combinations with replacement when we took k elements among n is $\binom{n+k-1}{k}$. Thus the number of call of function `Juxtapose` done by algorithm CUTJUXTAPOSE is $\prod_{i\in\{1,\ldots,T\}} \binom{n^{(i)}+m-2}{m-1}$. The time complexity of `Juxtapose` is $(m!)^{T-1}$ since all the permutations of the subsequences of type $t \neq 1$ are computed. Thus the time complexity of CUTJUXTAPOSE is $\left(\prod_{i\in\{1,\ldots,T\}} \binom{n^{(i)}+m-2}{m-1}\right)(m!)^{T-1}$, which is in $O(n^{(m-1)T}(m!)^{T-1})$. ■

If the number of machines, m, and the number of types, T, are constant, then the complexity of CUTJUXTAPOSE is $O(n^{T(m-1)})$: it is a polynomial time algorithm.

Note also that when there are only two types (A, B), `Juxtapose` reduces to finding the minimal cost bipartite matching between sets $\{J_1^A, \ldots, J_m^A\}$ and $\{J_1^B, \ldots, J_m^B\}$. The cost of matching J_i^A (with the total size $L_i^A = \sum_{j\in J_i^A} p_j$) with J_j^B (with total size L_j^B) is equal to $|J_i^A|\alpha_{B,A}L_j^B + |J_j^B|\alpha_{A,B}L_j^A$. By solving bipartite matching with the optimized Kuhn-Munkres algorithm [12], the complexity of `Juxtapose` is $O(m^3)$, and thus the complexity of the whole CUTJUXTAPOSE is $O(n^{(m-1)T}m^3)$.

IV. SPECIAL CASE: SINGLE TYPE

We consider in this section that all the tasks are of the same type, t_1. We start by proposing a polynomial time algorithm solving PSE in this case. This algorithm allows us to solve the special cases detailed in the introduction of Section III. Then, in Section IV-B we study general cost functions. So far, we studied a linear cost function: if all the tasks have the same type the cost of each task is proportional to the load of the machine on which it is. However, more complex cost functions are interesting from the systems perspective (e.g., webserver's response time in function of load is convex [4], [5], [6]). We show in Section IV-B that if the cost function is strictly convex then the problem becomes NP-hard in the strong sense. A concave cost function is not as realistic, since it would mean that, with unit size tasks, the average cost of a task decreases when the number of tasks on the same machine increases.

Algorithm 1: The CUTJUXTAPOSE algorithm

Notation:
`CR(n,m)` — generate all m tuples as combinations with replacements choosing m numbers from $1..n$; the returned tuples are sorted;
`Perm(m)` — generate all permutations of sequence $1..m$

`Juxtapose(` $(J_1^1, J_2^1, \ldots, J_m^1)$, $(J_1^2, J_2^2, \ldots, J_m^2)$, $\ldots$, $(J_1^T, J_2^T, \ldots, J_m^T)$`)`
  $cmin = \infty$; $optPart = []$;
  for π^2 *in* `Perm(m)` **do**
    for π^3 *in* `Perm(m)` **do**
      $\ddots$
        for π^T *in* `Perm(m)` **do**
          $part[1] \leftarrow J_1^1 \cup J_{\pi^2(1)}^2 \cup \cdots \cup J_{\pi^T(1)}^T$;
          $part[2] \leftarrow J_2^1 \cup J_{\pi^2(2)}^2 \cup \cdots \cup J_{\pi^T(2)}^T$;
          $\vdots$
          $part[m] \leftarrow J_m^1 \cup J_{\pi^2(m)}^2 \cup \cdots \cup J_{\pi^T(m)}^T$;
          if $C(Part) < cmin$ **then**
            $optPart \leftarrow part$;
            $cmin \leftarrow C(part)$;
  return $optPart$;

`CutJuxtapose(` $J^1, J^2, \ldots, J^T$`)`
  $cmin = \infty$; $optPart = []$;
  for $(k_1^1, k_2^1, \ldots, k_{m-1}^1)$ *in* `CR(`$n^{(1)}$,$m-1$`)` **do**
    for $(k_1^2, k_2^2, \ldots, k_{m-1}^2)$ *in* `CR(`$n^{(2)}$,$m-1$`)` **do**
      $\ddots$
        for $(k_1^T, k_2^T, \ldots, k_{m-1}^T)$ *in* `CR(`$n^{(T)}$,$m-1$`)` **do**
          $part \leftarrow$ `Juxtapose(`
          $((1, \ldots, k_1^1), (k_1^1+1, \ldots, k_2^1), \ldots, (k_{m-1}^1+1, \ldots, m))$,
          $\ldots$,
          $((1, \ldots, k_1^T), (k_1^T+1, \ldots, k_2^T), \ldots, (k_{m-1}^T+1, \ldots, m))$;
          if $C(part) < cmin$ **then**
            $optPart \leftarrow part$;
            $cmin = C(part)$;
  return optPart;

A. Optimal algorithm with linear costs

We show in this section a polynomial time algorithm, based on dynamic programming, for allocating tasks to machines. This algorithm is optimal for a single type and linear costs, thus finds the socially-optimal outcome in the Koutsoupias and Papadimitriou model [9], in which the cost of each task is the load of its machine.

For each machine $j \in \mathcal{M}$ we denote by L_j the load of machine j: $L_j = \sum_i$ on machine $_j\, p_i$. The cost of each task i is thus $c_i = L_{M_{P,i}}$, the load of the machine on which task i is allocated. As for a single type, the cost of a partition P is $\alpha_{t_1,t_1} \sum_{i=1}^n L_{M_{P,i}}$, without loss of generality, we fix $\alpha_{t_1,t_1} = 1$. We recall that the tasks are indexed in non-increasing order of loads: $p_1 \geq p_2 \geq \cdots \geq p_n$.

We denote by $C(x, r)$ the cost of an optimal solution of problem PSE when there are r machines and tasks of indices 1 to x ($x \in \{1, \ldots, n\}$, and $r \in \{1, \ldots, m\}$). The following dynamic programming algorithm (proved in Proposition IV-A.1) finds in polynomial time the optimal solution of PSE. The algorithm uses the SPT property (Section III-C): thus, when extending an allocation from r to $r+1$ machines, it checks allocations with $1, 2, \ldots, (x-1)$ smallest tasks on machine r. More formally, for all $x \in \{1, \ldots, n\}$, and $r \in \{1, \ldots, m-1\}$, we have:

$$C(x, r+1) = \min_{i \in \{1, \ldots, x-1\}} \left(C(x-i, r) + i \sum_{j=x-i+1}^{x} p_j \right). \tag{3}$$

The cost of an allocation on a single machine can be directly computed. For each $x \in \{1, \ldots, n\}$, we have:

$$C(x, 1) = x \sum_{i=1}^{x} p_i. \tag{4}$$

The optimal cost of a solution of PSE is $C(n, m)$. As usual, by backtracking we can deduce from $C(n, m)$ an allocation of minimal cost.

Proposition IV-A.1: The above dynamic programming algorithm optimally solves single type PSE in $O(n^2 m)$.

Proof: Let us first show that for each $x \in \{1, \ldots, n\}$ and $r \in \{1, \ldots, m\}$, the value $C(x, r)$ computed by the above dynamic programming is the optimal cost to allocate the x largest tasks of the considered instance on r machines. Once we will have shown this, we will deduce that $C(n, m)$ is the optimal cost of a solution of PSE, and thus that this algorithm is valid.

The proof is by induction on r, the number of machines. When there is only one machine, there is only one possible allocation, and its cost is equal to the number of tasks times the load of the machine. Thus Equation 4 is valid.

Let us now assume that for each $y \in \{1, \ldots, x-1\}$, $C(y, r)$ is the optimal cost to allocate the y largest task of the instance on r machines, and let us show that $C(x, r+1)$ is the optimal cost to allocate the x largest task of the instance on $r+1$ machines. Since there exists an optimal SPT allocation (as shown in Lemma III-C.1), there exists an optimal allocation $\mathcal{O}$ of the x largest tasks of the instance on $r+1$ machines, where the smallest tasks are on the same machine. In $\mathcal{O}$, let $i^* \in \{1, \ldots, n\}$ be the number of tasks which are on the machine to which the smallest task x is allocated. As $\mathcal{O}$ is an SPT allocation, this machine has the i^* smallest tasks $x - i^* + 1, \ldots, x$. The cost of $\mathcal{O}$ is $C(x - i^*, r) + i^* \sum_{j=x-i^*+1}^{x} p_j$. Indeed, $\sum_{j=x-i^*+1}^{x} p_j$ is the cost of each of the i^* smallest tasks in $\mathcal{O}$ and $C(x - i^*, r)$ is by induction the minimal cost to allocate the other tasks (the $x - i^*$ largest tasks on r machines). Equation 3 computes the cost of a feasible solution: if $i' \in \{1, \ldots, n\}$ is the value of i that minimizes $\left(C(x-i, r) + i \sum_{j=x-i+1}^{x} p_j\right)$ then this equation computes the cost of a solution where the i' smallest tasks are on the same machine, and the other tasks are partitionned optimally on the r remaining machines. This value is minimized when $i' = i^*$. Thus $C(x, r+1)$ is the

minimum cost of a partition of the x largest tasks on $r+1$ machines.

Therefore, $C(n,m)$ is the cost of an optimal solution of problem PSE. It can be be computed in $O(n^2m)$. Indeed, we stock the values $C(x,r)$ on a $n \times m$ matrix. Each value $C(x,r+1)$ can be computed in $O(n)$ once the values $C(x-i,r)$ are known (for each $x \in \{2,\ldots,n\}$, $\sum_{j=2}^{x} p_j$ can be computed at the beginning of the algorithm in $O(n)$, and then when we compute $C(x,r+1)$ we have $x \leq n$ costs to examine — each cost being computed in $O(1)$ if we start by $i = x-1$ and decrement i until $i = 1$). ■

B. Complexity for strictly convex cost

Until now, we assumed that a task's cost is a linear combination of the loads of the different types on its machine. Many phenomena are, however, not linear. For instance, a webserver's query response time rises slowly with load until a certain threshold, but then it rapidly increases [4], [5], [6]. In this section, we show that it is NP-hard to minimize costs given by a convex cost function, even for a single type.

More formally, we assume in this section that the cost of task i in partition P is $c_i = f(L_{M_{P,i}})$, where f is a strictly convex function (and $L_{M_{P,i}}$ is the load of the machine on which task i is allocated in P). Let us denote by PCSE (which stands for "Partition with Convex Side Effects") the following problem (note that this problem has a natural extension to several types):
Input: n tasks (of different sizes), a number m of machines, and an increasing and strictly convex cost function f.
Output: a partition which minimizes the sum of the costs $\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} f(L_{M_{P,i}})$.

Proposition IV-B.1: For any cost function f which is increasing and strictly convex, the decision version of problem PCSE is strongly NP-complete.

Proof: We do a reduction from problem 3-PARTITION, which is strongly NP-complete [11], to our problem. The 3-PARTITION problem is the following one: the input is a finite set A of $3q$ elements, a bound $B \in \mathbb{Z}^+$, and a "size" $s(a) \in \mathbb{Z}^+$ for each $a \in A$, such that $s(a)$ satisfies $\frac{B}{4} < s(a) < \frac{B}{2}$ and such that $\sum_{a \in A} s(a) = qB$; the question is: can A be partitioned into q disjoint sets $S_1,\ldots,S_q$ such that, for $i \in \{1,\ldots,q\}, \sum_{a \in S_i} s(a) = B$?

The decision version of problem PCSE is as follows: given an instance of problem PCSE, is there a partition P with total cost $C(P) = \sum_{i=1}^{n} c_i$ at most K ?

The instance of PCSE corresponding to an instance of the 3-PARTITION is the following: we have $m = q$ machines $\{M_1,\ldots,M_m\}$ and $n = 3q$ tasks. For each element $a \in A$, we have a task j_a of size $p_a = s(a)$. We set the budget $K = nf(B)$.

Let us now show that there is a solution to the 3-PARTITION problem if and only if there is a solution to the corresponding instance of the PCSE problem. Assume first that there is a solution to the 3-PARTITION problem. Let $\{S_1,\ldots,S_m\}$ be the sets obtained in this solution. Let us consider the partition where, for each $i \in \{1,\ldots,m\}$, the tasks which correspond to the elements of S_i are allocated on M_i. In this partition, since $\sum_{a \in S_i} s(a) = B$, the load on each machine is equal to B. The cost of each task is thus $f(B)$, and the sum of the costs is $nf(B)$. There is a solution of cost $K = nf(B)$ for the PCSE problem.

Let us now assume that there is a solution P to the PCSE problem, and let us show that there is a solution to the corresponding instance of the 3-PARTITION problem. Let n_j be the number of tasks on M_j in P, and let L_j be the load of M_j in P. The cost of P is $C(P) = \sum_{k=1}^{m} n_k f(L_k)$. Assume by contradiction that there is in P a machine M_i such that $n_i > 3$. In this case, there exists in P a machine M_j such that $n_j < 3$ (since there are $3m$ tasks). On M_i, the load is $L_i > B$, since there are at least 4 tasks on M_i and each task has a size strictly larger than $\frac{B}{4}$. On the contrary, on M_j, the load is $L_j < B$, since there are at most 2 tasks on M_j and each task has a size strictly smaller than $\frac{B}{2}$. Thus, $n_i > n_j$ and $L_i > L_j$ (and thus $f(L_i) > f(L_j)$). Hence $n_i f(L_i) + n_j f(L_j) > (n_i - 1)f(L_i) + (n_j + 1)f(L_j)$. By increasing each n_j smaller than 3 up to 3, and by decreasing each n_i larger than 3 up to 3, we decrease at each step the cost of the solution (each time we increase by one unit the coefficient of $f(L_j)$, we simultaneously decrease by one unit the coefficient of $f(L_i) > f(L_j)$). We obtain that $C(P) = \sum_{k=1}^{m} n_k f(L_k) > \sum_{k=1}^{m} 3f(L_k)$. Since f is strictly convex, $\sum_{k=1}^{m} 3f(L_k) \geq 3mf(\frac{\sum_{k=1}^{m} L_k}{m}) = 3mf(B) = K$. Thus $C(P) > K$: there is a contradiction. Thus there are exactly three tasks per machine in P.

Since there are three tasks per machine, and since P is a solution of problem PCSE, we have $C(P) = 3\sum_{i=1}^{m} f(L_i) \leq K = 3mf(B)$. Since $B = \frac{\sum_{i=1}^{m} L_i}{m}$, $\sum_{i=1}^{m} f(L_i) \leq mf(\frac{\sum_{i=1}^{m} L_i}{m})$. Since f is strictly convex, this can be true only if for each $i \in \{1,\ldots,m\}$, $L_i = \frac{\sum_{i=1}^{m} L_i}{m}$. In this case, there are three tasks per machine, and the load of each machine is B. If we denote by S_i the set of elements of A corresponding to the tasks allocated on M_i in P, we have for each set S_i, with $i \in \{1,\ldots,m\}$, $\sum_{a \in S_i} s(a) = B$: there is a solution to the 3-PARTITION problem.

There is a solution to the 3-PARTITION if and only if there is a solution to the corresponding instance of the PCSE. As PCSE is (trivially) in NP, PCSE is thus strongly NP-complete. ■

V. RELATED WORK

Alternative models of data center resource management. There is no standard model of data center resource management (standard in the sense in which the parallel job model is standard for HPC). Existing models can be roughly categorized into variants of multi-dimensional bin-packing (to model heterogeneous resource requirements),

stochastic optimization (to model uncertainty), and statistical approaches.

Multi-dimentional bin-packing. In bin-packing approaches, tasks are modeled as items to be packed into bins (machines) of known capacity [13]. To model heterogeneous tasks and resources, bin packing is extended to vector packing: an item's size is defined as a vector with dimensions corresponding to requirements on individual resources (CPU, memory, disk or network bandwidth) [14]. These are hard optimization problems: bin packing is strongly NP-hard (but has an asymptotic PTAS [15]), while two-dimensional vector packing does not admit an asymptotic PTAS [16] Alternatively, if tasks have unit-size requirements, simpler representations can be used, such as maximum weighted matching [17]. In our model, machines' capacities are not crisp—instead, tasks' costs gradually increase with increased load.

Stochastic versions of combinatorial optimization problems. Stochastic versions of classical optimization problems [18], [19], [20], [21] can be used to model uncertainty of tasks' resource requirements or their variability in time. In these representations, some parameters of an instance are random variables, e.g., items' sizes in bin packing. A typical goal is to construct an optimal solution (in terms of the number of bins used, or the value of the items picked to a knapsack) that violates the capacity constraints only with a small probability. These models, however, rarely lead to practical algorithms, at the same time requiring restrictive assumptions on the stochastic models of jobs, as usually the algorithms work only for a certain distribution.

Statistical approaches. Bobroff et al. [22] uses statistics of the past CPU load of tasks (CDF, autocorrelation, periodograms) to predict the load in the "next" time period; then they use bin packing to calculate a partition minimizing the number of used bins subject to a constraint on the probability of overloading servers. Di et al. [23] analyze resource sharing for streams of tasks to be processed by virtual machines. Sequential and parallel task streams are considered in two scenarios—when there are sufficient resources to run all tasks; and when the resources are insufficient. For sufficient resources, optimality conditions are formulated; for insufficient resources, fair scheduling policies are proposed.

Analysis of effects of colocation. Podzimek et al. [8] analyze the performance of colocated CPU-intensive tasks. Their measured performance interference metric is similar to our α_{t_j,t_i} coefficient. Kim et al. [7] focuses on experimental measures of performance interference between a few concrete HPC applications. This interference, called the affinity metric, is similar to our α_{t_j,t_i} coefficients. They propose a greedy allocation heuristics, but they don't study the complexity of the problem, nor the optimality of their heuristics.

Game-theoretic approaches. There is a strong connection between our model and games, in which each task is owned by a selfish agent who wants to minimize task's cost.

Load balancing games. Our model relates to the load-balancing games introduced by Koutsoupias and Papadimitriou [9], in which the cost of each task is the total load of the machine to which the task is allocated. This model represents, e.g., a system of servers from which users download large files: tasks correspond to requests of individual users and each the user aims at contacting a server with the smallest load [10]. Contrarily to what we do in this paper, the game model considers that each task is owned by an agent, and that each agent chooses on which machine its task will be scheduled. In most papers, the authors aim at minimizing the maximum load over all the machines (see [10] for a survey), but in some papers [24], [25] the aim is to minimize the average social cost, as we do in our paper. However, to our best knowledge, no centralized optimal algorithm to minimize the average cost of the tasks has been studied. In Section IV-A, we have given a polynomial time algorithm which solves this problem.

Coalition structure generation. Our model is also related to the coalition structure generation (CSG, see [26] for a recent overview). CSG consists in partitioning a set of agents (tasks) into subsets (called coalitions); each agent affects the cost of the coalition she is assigned to (but not the costs of the other coalitions). However, in CSG the aim is to minimize the total cost of all coalitions (and not the average cost of an agent). Additionally, in CSG the number of coalitions is not bounded, while we bound the number of subsets by the number of machines m. Aziz et al. [27] analyzes CSG with players having types. When the number of types is a constant, they give a polynomial algorithm. However, their notion of type is more restrictive than ours: two players have the same type if their influence on the costs of the others is exactly the same.

VI. Conclusions and Perspectives

We propose a new model describing the performance of tasks colocated on machines. Our model introduces the notion of *type*. Types describe and allow to deal with tasks' heterogeneity: e.g., a computationally-intensive task and a database instance influence the performance of a web server in a different way. We studied Partition with Side Effects, a family of problems in which the cost of a task is a function of the loads of different types allocated to the same machine. For linear cost functions, we showed that the problem is strongly NP-complete for an arbitrary number of types. We proposed two optimization algorithms: one polynomial in the number of tasks and machines (but exponential in the number of sizes of tasks and types); and another one polynomial in the number of tasks (but exponential in the number of machines and types). For a single type, we gave a

polynomial time algorithm. We also proved that the problem becomes strongly NP-complete when the cost function is strictly convex, and this even for a single type.

We leave open the NP-completeness of Partition with Side Effects with a constant number of types.

A systems-oriented future work is to validate our model in an actual data center resource manager: this would open many interesting questions on, e.g., automatic classificiation of tasks into types or inferring their coefficients.

In this paper, we minimized the total cost. Another natural research direction is to add weights to the tasks (and to minimize the weighted total cost), or to minimize the maximal cost. Note that even for a single type and two machines this last problem is NP-hard since it reduces to the widely studied optimization of the makespan on parallel machines $(P||C_{max})$.

The notion of type of a task can also be applied to generalize other problems, as for example load balancing games (load balancing games with types would correspond to our problem in which agents choose themself on which machines their tasks will be scheduled). On an opposite side, our notion of types may be also used to reduce the complexity of coalition structure generation problems.

Acknowledgements: We thank the anonymous reviewers for their helpful comments.
This research has been partly supported by Polish National Science Center grant Sonata (UMO-2012/07/D/ST6/02440) and by the Google Faculty Research Award.

REFERENCES

[1] D. Klusácek and H. Rudová, "Multi-resource aware fairsharing for heterogeneous systems," in *JSSPP, Proc.*, 2014.

[2] S. Di, D. Kondo, and F. Cappello, "Characterizing and modeling cloud applications/jobs on a Google data center," *The Journal of Supercomputing*, vol. 69, no. 1, pp. 139–160, 2014.

[3] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *SoCC, Proc.* ACM, 2012, p. 7.

[4] L. P. Slothouber, "A model of web server performance," in *WWW, Proc.*, 1996.

[5] J. Cao, M. Andersson, C. Nyberg, and M. Kihl, "Web server performance modeling using an m/g/1/k* PS queue," in *ICT, Proc.*, vol. 2. IEEE, 2003, pp. 1501–1506.

[6] G. Khanna, K. Beaty, G. Kar, and A. Kochut, "Application performance management in virtualized server environments," in *NOMS, Proc.* IEEE, 2006, pp. 373–381.

[7] S. Kim, E. Hwang, T.-K. Yoo, J.-S. Kim, S. Hwang, and Y.-R. Choi, "Platform and co-runner affinities for many-task applications in distributed computing platforms," in *CCGrid Proc.* IEEE CS, 2015.

[8] A. Podzimek, L. Bulej, L. Y. Chen, W. Binder, and P. Tuma, "Analyzing the impact of cpu pinning and partial cpu loads on performance and energy efficiency," in *CCGrid Proc.* IEEE CS, 2015.

[9] E. Koutsoupias and C. Papadimitriou, "Worst-case equilibria," in *STACS*, ser. LNCS, C. Meinel and S. Tison, Eds. Springer, Jan. 1999, no. 1563, pp. 404–413.

[10] B. Vöcking, "Selfish load balancing," in *Algorithmic Game Theory*, N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, Eds. Cambridge University Press, Sep. 2007, pp. 517–542.

[11] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, Jan. 1979.

[12] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *JACM*, vol. 19, no. 2, pp. 248–264, 1972.

[13] E. G. Coffman Jr, M. R. Garey, and D. S. Johnson, "Approximation algorithms for bin packing: A survey," in *Approximation algorithms for NP-hard problems*, D. Hochbaum, Ed. PWS, 1996, pp. 46–93.

[14] M. Stillwell, F. Vivien, and H. Casanova, "Virtual machine resource allocation for service hosting on heterogeneous distributed platforms," in *IPDPS Procs.* IEEE, 2012, pp. 786–797.

[15] W. Fernandez de la Vega and G. Lueker, "Bin packing can be solved within 1 + epsilon in linear time," *Combinatorica*, vol. 1, no. 4, pp. 349–355, 1981.

[16] G. Woeginger, "There is no asymptotic PTAS for two-dimensional vector packing," *Information Processing Letters*, vol. 64, no. 6, pp. 293–297, 1997.

[17] O. Beaumont, L. Eyraud-Dubois, C. Thraves Caro, and H. Rejeb, "Heterogeneous resource allocation under degree constraints," *IEEE TPDS*, vol. 24, no. 5, pp. 926–937, 2013.

[18] K. W. Ross and D. H. Tsang, "The stochastic knapsack problem," *Communications, IEEE Trans. on*, vol. 37, no. 7, pp. 740–747, 1989.

[19] A. Goel and P. Indyk, "Stochastic load balancing and related problems," in *FOCS*. IEEE, 1999, pp. 579–586.

[20] M. Chen, H. Zhang, Y.-Y. Su, X. Wang, G. Jiang, and K. Yoshihira, "Effective vm sizing in virtualized data centers," in *IFIP/IEEE IM*. IEEE, 2011, pp. 594–601.

[21] M. Wang, X. Meng, and L. Zhang, "Consolidating virtual machines with dynamic bandwidth demand in data centers," in *INFOCOM, Proc.* IEEE, 2011, pp. 71–75.

[22] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing SLA violations," in *IM, Proc.* IEEE, 2007, pp. 119–128.

[23] S. Di, D. Kondo, and C. Wang, "Optimization of composite cloud service processing with virtual machines," *IEEE Trans. on Computers*, 2015 (in print).

[24] B. Awerbuch, Y. Azar, and A. Epstein, "The price of routing unsplittable flow," in *STOC, Proc.*, 2005, pp. 57–66.

[25] G. Christodoulou and E. Koutsoupias, "The price of anarchy of finite congestion games," in *STOC, Proc.*, 2005, pp. 67–73.

[26] E. Elkind, T. Rahwan, and N. R. Jennings, "Computational coalition formation," in *Multiagent Systems*, G. Weiss, Ed. MIT Press, 2013.

[27] H. Aziz and B. De Keijzer, "Complexity of coalition structure generation," in *AAMAS, Proc.*, 2011, pp. 191–198.