Optimizing egalitarian performance when colocating tasks with types for cloud data center resource management

Fanny Pascual, Krzysztof Rzadca *Member, IEEE,*

Abstract—In data centers, up to dozens of tasks are colocated on a single physical machine. Machines are used more efficiently, but the performance of the tasks deteriorates, as the colocated tasks compete for shared resources. Since the tasks are heterogeneous, the resulting performance dependencies are complex. In our previous work [26], [27] we proposed a new combinatorial optimization model that uses two parameters of a task — its size and its type — to characterize how a task influences the performance of other tasks allocated to the same machine.

In this paper, we study the egalitarian optimization goal: the aim is to optimize the performance of the worst-off task. This problem generalizes the classic makespan minimization on multiple processors ($P||C_{\max}$). We prove that polynomially-solvable variants of $P||C_{\max}$ are NP-hard for this generalization, and that the problem is hard to approximate when the number of types is not constant. For a constant number of types, we propose a PTAS, a fast approximation algorithm, and a series of heuristics. We simulate the algorithms on instances derived from a trace of one of Google clusters. Compared with baseline algorithms solving $P||C_{\max}$, our proposed algorithms aware of the types of the jobs lead to significantly better tasks' performance.

The notion of type enables us to extend standard combinatorial optimization methods to handle degradation of performance caused by colocation. Types add a layer of additional complexity. However, our results — approximation algorithms and good average-case performance — show that types can be handled efficiently.

Index Terms—cloud computing; scheduling; complexity; approximation algorithm; heterogeneity; co-tenancy; workload co-location

1 Introduction

A modern cloud data center redefines the way the industry and the academia compute. Resource management in data centers significantly differs from scheduling jobs on typical High Performance Computing (HPC) supercomputers. First, the workload is much more varied [5], [30]. Indeed, data centers act as a physical infrastructure providing virtual machines, or higher-level services, such as memory-cached databases or network-intensive web applications. In contrast, there are relatively few HPC-like computationally-intensive batch jobs (we will use a generic term *task* for all these categories). Thus, in a data center, a task usually does not saturate the resources of a single node [19]. Second, the loads of the tasks vastly differ: in a published trace [30], tasks' average CPU loads span more than 4 orders of magnitude. Therefore, in contrast to HPC scheduling in which jobs rarely share a node, multiple tasks are commonly allocated to the same physical machine. Finally, certain classes of data center tasks, such as web servers or databases, almost persistently serve user traffic and never "complete". Thus, in contrast to scheduling in HPC, the aim is not to complete such tasks as soon as possible, but rather to allocate sufficient resources so that its perceived performance is satisfactory.

Tasks colocated on a machine compete for shared hardware. Despite significant advances in both OS-level fairness and VM hypervisors, virtualization is not transparent: multiple studies show [19], [20], [21], [29], [36] that the performance of colocated tasks drops. Suspected reasons include difficulties in sharing the CPU cache or the memory bandwidth. In order to optimize the performance of the tasks, the resource manager should thus colocate tasks that are compatible, i.e., that use different kinds of resources. This, however, requires a performance model.

Typical approaches to colocating tasks on machines rely on bin-packing [10], [16], [25], [31], [32], [33] thus they implicitly assume a crisp performance model — as long as the total demand of colocated tasks remains below the resources available on a machine, the tasks' performance is considered satisfactory. Complex inter-tasks performance degradation have to be modeled by placement constraints [10], [25].

In contrast, in our *side-effects model* [26], [27] rather than trying to predict tasks' performance from OS-level metrics, or ignoring it, we derive it from two characteristics of each task: task's *type* (e.g.: a database, or a computationally-intensive job) and task's *size* relative to other tasks of the same type (e.g.: number of requests per second). The total load of a machine is a vector: its i-th dimension is the sum of sizes of tasks of the i-th type placed on this machine. Each type additionally defines a performance function mapping this vector of loads to a type-relevant performance metric. As datacenters execute multiple instances of tasks, such a function can be inferred by a monitoring module [20], [29], [36] which will match task's reported performance (such as the 95th percentile response time) with observed or reported loads.

We use a linear performance function: on each machine, the

- *F. Pascual is at Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6, F-75005 Paris, France. Email: fanny.pascual@lip6.fr.*
- *K. Rzadca is at Institute of Informatics, University of Warsaw, Poland Email: krz@mimuw.edu.pl.*

A preliminary version of this paper was published as F. Pascual, K. Rzadca, Optimizing egalitarian performance in the side-effects model of colocation for data center resource management, in F.F. Rivera et al. (Eds.): Euro-Par 2017 Proceedings, LNCS 10417, Springer.

influence that a type t' has on the performance of a task of type t is a product of the load of type t' on this machine and a coefficient $\alpha_{t',t}$. The coefficient $\alpha_{t',t}$ describes how compatible t' load is with a task of type t (our coefficients are similar to interference/affinity metrics proposed in [20], [29]). Low values ($0 \leq \alpha_{t',t} < 1$) correspond to compatible types (e.g. colocating a memory-intensive and a CPU-intensive task): it is preferable to colocate a task t with tasks of the other type t', rather than with other tasks of its own type t. For yet another type t'', high values ($\alpha_{t'',t} > 1$) denote types competing for resources.

In this paper, we study the egalitarian objective — we minimize *the maximum* of the tasks' costs (in our previous works [26], [27] we studied the utilitarian objective — we minimized *the sum* of the tasks' costs). The egalitarian objective corresponds to maximizing the quality of service proposed to all tasks. When there is only one type, or when for all the types t, t' our coefficients are $\alpha_{t,t'} = 1$, the cost of a task is the load of the machine on which it is assigned and our problem is equivalent to the classical multiprocessor scheduling, $P||C_{\max}$ (minimization of the makespan, i.e., the maximum load).

The contributions of this paper are as follows.

1) We prove that the notion of type adds complexity, as makespan minimization with unit tasks $P|p_i = 1|C_{\max}$, a polynomially solvable variant of $P||C_{\max}$, becomes NP-hard when types are considered. Moreover, the problem becomes hard to approximate when the number of types T is not constant. We then show how to cope with this added complexity.
2) We propose a polynomial time approximation scheme (PTAS) for a constant T and constant values of the coefficients α.
3) We also provide a fast greedy approximation algorithm.
4) For a qualitative understanding of how the coefficients α shape the optimal allocation, we study a special case where there are two types. We identify two tipping points, i.e., values of α for which the optimal allocation radically changes. For each of the three induced cases, we give a fast approximation algorithm.
5) We test our algorithms by simulation on a trace derived from one of Google clusters. The simulations confirm excellent average-case performance of our type-aware algorithms.

The paper has the following organization. In Section 2 we formally define our resource management model. In Section 3 we demonstrate that our problem is NP-hard and hard to approximate within a constant factor when the number of types is non-constant. In Section 4 we propose two approximation algorithms for a fixed number of types: a PTAS, mostly of theoretical interest; and a fast approximation called FILLGREEDY. In Section 5 we propose four alternative heuristic algorithms. In Section 6 we analyze a special case of the problem with two types for which we prove approximation ratios for some of the heuristics. In Section 7, we evaluate the algorithms by simulation. Finally, In Section 8, we discuss related work.

2 SIDE-EFFECTS OF COLOCATING TASKS: A MODEL

We consider a system that allocates n independent tasks $J = \{1, \ldots, n\}$ to m identical machines $\mathscr{M} = \{M_1, \ldots, M_m\}$. Each task i has a known size $p_i \in \mathbb{N}$ (our model is clairvoyant, a common assumption in scheduling; the sizes can be estimated from previous submissions or users' estimates). The size corresponds to the load the task imposes on a machine: the request rate for a web server; or the CPU load for a CPU-intensive computation. We take other assumptions standard in scheduling theory: all the tasks are known (off-line) and ready to be assigned to a machine (released at time 0). We take these assumptions to derive results on the basic model before tackling more complex ones. We denote by $p_{\max} = \max_{i \in \{1,\ldots,n\}} p_i$ the largest size and by W the total load, $W = \sum_{i=1}^{n} p_i$. [1] We assume that the tasks are indexed by non-increasing sizes: $p_1 \geq p_2 \geq \cdots \geq p_n$.

A *partition* (an *allocation*) is an assignment of each of the n tasks to one of the m machines. A partition separates the tasks into at most m subsets: each subset corresponds to the tasks allocated on the same machine. Given a partition P, we denote by $M_{P,i} \in \mathscr{M}$ the machine on which task i is allocated. Due to the similarities with $P||C_{max}$, we sometimes use the term "schedule" (and the symbol σ) for an allocation (and even the term of length for the size of a task). In this case, only the allocation is meaningful (not the order of the tasks on the machines).

The impact of task i on the performance 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. The operator of the data center should define types according to observed performance dependencies. A type could correspond to a specific application (as in [20]); but it could also be more general, gathering, for example, all webservers under a single type, and all databases under another one. Here we assume that the task's type is known to the resource manager either from the analysis of previous submissions, or from users' declarations (this assumption corresponds to the clairvoyance assumption in classic scheduling). Let $\mathscr{T} = \{1, \ldots, T\}$ be a set of T different types of tasks. We denote by $t_i \in \mathscr{T}$ the type of task i. For each type $t \in \mathscr{T}$, we denote by $J^{(t)}$ the set of the tasks which are of type t. We denote by $p_i^{(t)}$ the size of the i-th largest task of type t (ties are broken arbitrarily).

We express performance of a task i by a cost function c_i: to simplify presentation of our results, we prefer to express our problems as minimization of costs, rather than maximization of performance (for a single type, our cost is synonymous with the makespan). Note that the cost is unrelated to monetary cost (the amount of money that a job pays to the machine) — we do not consider monetary costs in this paper. The cost c_i of task i depends on the *total load* of tasks j colocated on the same machine $M_{P,i}$, but different types have different impacts:

$$c_i = \sum_{j \text{ on machine } M_{P,i}} p_j \cdot \alpha_{t_j,t_i} \tag{1}$$

Note that the cost function takes into account the task i itself, as well as the other tasks of the same type. A coefficient $\alpha_{t,t'} \in \mathbb{R}_{\geq 0}$ defined for each pair of types $(t,t') \in \mathscr{T}^2$, 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 necessarily symmetric, i.e., it is possible that $\alpha_{t,t'} \neq \alpha_{t',t}$. The coefficients $\alpha_{t,t'}$ can be estimated by monitoring tasks' performance as a function of their colocation and their sizes (a data center runs many instances of similar services). Previous works [20], [29], [36] and to some extent [37] show how to form a performance model as a function

[1] To compute the cost of allocation, our model weights the loads by coefficients. We define W as a simple sum of loads of tasks (perhaps of different types) and $p_{\max}$ as a simple maximum; however, this does not lead to inconsistencies, as we use these values only as bounds.

TABLE 1
Table of notations.

c_i	cost of task i (as defined in Equation 1)
J	set of the tasks
$J^{(t)}$	set of the tasks of type t
m	number of machines
M_i	i-th machine
$\mathscr{M}$	set of machines
n	number of tasks
p_i	size of the i-th largest task
$p_i^{(t)}$	size of the i-th largest task of type t
$p_{\max}$	largest size of a task: $p_{\max} = \max_{i \in \{1,\dots,n\}} p_i = p_1$
T	number of types
t_i	type of task i
$\mathscr{T}$	set of the types
W	total load : $W = \sum_{i=1}^{n} p_i$
$W^{(t)}$	total load of tasks of type t: $W^{(t)} = \sum_{t_i = t} p_i$
$\alpha_{t,t'}$	coefficient which measures the impact of a unit of task of type t on the cost of a task of type t'
$\alpha_{\max}$	maximal value of the coefficients: $\alpha_{\max} = \max_{(t,t') \in \mathscr{T}^2} \alpha_{t,t'}$
α	coefficient $\alpha_{t,t'}$ when we consider only two types t and t' ($T = 2$)

of colocation—we discuss the similarities in Section 8. In this paper, following the assumption of clairvoyance, common in scheduling, we assume that the coefficients $\alpha_{t,t'}$ are given.

We consider the linear cost function which generalizes, by adding coefficients $\alpha_{t,t'}$, the fundamental scheduling problem $P||C_{\max}$ [13]. Assuming linearity is a common approach when constructing models in operational research or statistics (e.g. linear regressions). Likewise, in selfish load balancing games [22], [35], it is assumed that the cost of each task is the total load of the machine (their model does not consider types). We assume that the impact the type has on itself is *normalized* with regards to tasks' sizes, i.e., $\alpha_{t,t} = 1$ (although some of our results, notably the PTAS, do not need this assumption). We denote maximal coefficient by $\alpha_{\max} = \max_{(t,t') \in \mathscr{T}^2} \alpha_{t,t'}$.

We denote by MCT (MINMAXCOST WITH TYPES) the problem of finding a partition P^* minimizing the maximum cost $C(P) = \max_{i \in \{1,\dots,n\}} c_i$. The partition P^* minimizes the worst performance a task experiences in the system, and thus corresponds to the egalitarian fairness.

Notations which are used through the paper are summarized in Table 1 (note that the table purposely skips notation used for, e.g., a single proof).

3 COMPLEXITY AND HARDNESS OF MCT FOR T NOT FIXED

MCT is NP-hard as it generalizes the NP-hard problem $P||C_{\max}$ when there is only one type. We first show that a polynomially-solvable variant of multiprocessor scheduling ($P|p_i = 1|C_{\max}$) becomes NP-hard when tasks are of different types. Thus types add another level of complexity onto an already NP-hard problem.

Proposition 1. *The decision version of* MCT *is NP-complete, even if all the tasks have unit size, and even if there are only two machines.*

Proof. We reduce from the NP-complete PARTITION problem [11]. In PARTITION the input is a finite set $A = \{a_1, \dots, a_k\}$ of k positive integers summing up to $2B$ ($\sum_{a \in A} a = 2B$). The question is: can A be partitioned into two disjoint sets A_1, A_2 such that $\sum_{a \in A_1} a = \sum_{a \in A_2} a = B$?

The instance of MCT corresponding to the instance of PARTITION is as follows. We have 2 machines $\{M_1, M_2\}$, $k+2$ types, and $k+2$ tasks $\{1, \dots, k+2\}$. Each task is of size 1, and each of a different type (for all $i \in \{1, \dots, k+2\}$, type of task i will be i). For each type $i \in \{1, \dots, k\}$, and for each type $j \in \{1, \dots, k+2\}$, with $i \neq j$, we set $\alpha_{i,j} = a_i$ (we have $\alpha_{i,i} = 1$). We have $\alpha_{k+1,k+2} = \alpha_{k+2,k+1} = 2B$ (and $\alpha_{k+1,k+1} = \alpha_{k+2,k+2} = 1$). For each type $i \in \{1, \dots, k\}$, we have $\alpha_{k+1,i} = \alpha_{k+2,i} = a_i$.

Let us now show that the answer of the PARTITION problem is "yes" if and only if, in the corresponding instance of MCT, there exists an allocation P with maximal cost at most $B+1$.

Let us first assume that there is a partition (A_1, A_2) of A. For each integer $i \in \{1, \dots, k\}$, if $a_i \in A_1$ (respectively $a_i \in A_2$), then we put task i on machine M_1 (respectively M_2). Task $k+1$ (resp. $k+2$) is on machine M_1 (resp. M_2). We first show that in this solution the cost of each task on M_1 is $B+1$. The cost of task $k+1$ is $\alpha_{k+1,k+1} + \sum_{j \text{ on } M_1, j \neq k+1} \alpha_{j,k+1} = 1 + \sum_{j \text{ on } M_1, j \neq k+1} a_j = 1 + \sum_{a_j \in A_1} a_j = B+1$. The cost of each task $i \neq k+1$ on M_1 is $\alpha_{i,i} + \alpha_{k+1,i} + \sum_{j \text{ on } M_1, j \notin \{i,k+1\}} \alpha_{j,i} = 1 + a_i + \sum_{j \text{ on } M_1, j \notin \{i,k+1\}} a_j = 1 + \sum_{a_j \in A_1} a_j = B+1$. Likewise, the cost of each task on M_2 is $B+1$. Therefore, there is a solution of maximal cost at most $B+1$ for problem MCT.

Let us now assume that there is a solution P of maximal cost at most $B+1$ for MCT. We know that tasks $k+1$ and $k+2$ are not on the same machine in P (otherwise the maximum cost would be at least $2B+1$). Let us assume without loss of generality that task $k+1$ is on M_1 and task $k+2$ is on M_2. Let A_1 be the set of numbers of A which corresponds to tasks of $\{1, \dots, k\}$ on M_1 in P. Likewise, let A_2 be the set of numbers of A which corresponds to tasks of $\{1, \dots, k\}$ on M_2 in P. We first consider the tasks assigned to M_1. The cost of task $k+1$ is $\alpha_{k+1,k+1} + \sum_{j \text{ on } M_1 | j \in \{1,\dots,k\}} \alpha_{j,k+1} = 1 + \sum_{j \text{ on } M_1 | j \in \{1,\dots,k\}} a_j = 1 + \sum_{a_j \in A_1} a_j$. The cost of task $i \neq k+1$ on M_1 is $\alpha_{i,i} + \alpha_{k+1,i} + \sum_{j \text{ on } M_1 | j \in \{1,\dots,k\}, j \neq i} \alpha_{j,i} = 1 + a_i + \sum_{j \text{ on } M_1 | j \in \{1,\dots,k\}, j \neq i} a_j = 1 + \sum_{j \text{ on } M_1 | j \in \{1,\dots,k\}} a_j = 1 + \sum_{a_j \in A_1} a_j$. Hence, each task assigned to M_1 has a cost equal to $1 + \sum_{a_j \in A_1} a_j$.
The value of $\sum_{a_j \in A_1} a_j$ is at most B, since P is a solution of maximal cost at most $B+1$. Likewise, we can show that each task on M_2 has a cost equal to $1 + \sum_{a_j \in A_2} a_j$, and we know that this cost is at most $B+1$. Therefore, we get that $\sum_{a_j \in A_2} a_j \leq B$. Since $\sum_{a_j \in A_1} a_j + \sum_{a_j \in A_2} a_j = 2B$, we get that $\sum_{a_j \in A_1} a_j = \sum_{a_j \in A_1} a_j = B$: the answer to the PARTITION problem is "yes". □

We now show that if the number of types is not fixed, MCT is hard to approximate within a constant factor, even if all tasks have the same size (again, in contrast to polynomially-solvable $P|p_i = 1|C_{\max}$).

Proposition 2. MCT *is strongly NP-hard, even if all tasks have unit size. Moreover, there is no polynomial time r-approximate algorithm for* MCT*, for any number $r > 1$, unless $P = NP$.*

Proof. We show that a r-approximate algorithm for MCT would solve the NP-complete PARTITION INTO CLIQUES, PIC [11]. In PIC, the input is a graph $G = (V, E)$ and a positive integer $K \leq |V|$ (we assume that V are labeled from 1 to $|V|$). The question is whether the vertices of G can be partitioned into $k \leq K$ disjoint sets $V_1, V_2, \dots, V_k$ such that, for $1 \leq i \leq k$, the subgraph induced by V_i is a complete graph.

Given an instance of PIC, we create the following instance of MCT. The number of machines is $m = K$. There are $n = |V|$ tasks $\{1, \dots, n\}$ (a task corresponds to a node of the graph). Each task is of a different type. Types are labeled from 1 to $|V|$: the type of task i is i. All the tasks are of size 1. For each type i, $\alpha_{i,i} = 1$. The set of edges of G corresponds to 0 cost coefficients between the corresponding tasks: for each pair of types $(i, j), i \neq j$: $\alpha_{i,j} = 0$ if $\{i, j\} \in E$ and $\alpha_{i,j} = r$ if $\{i, j\} \notin E$.

We claim that a solution of the MCT instance costs either 1 or at least $r+1$. We also claim that the answer for the instance of PIC is "yes" if and only if the optimal cost of the MCT instance is 1. If a solution of cost 1 exists, an r-approximate algorithm has to return a solution of cost at most r. As all other solutions cost at least $r+1$, the r-approximate algorithm has to return a solution of cost 1. Since $K \leq |V|$, if we assume that the r-approximate algorithm runs in polynomial time, then it solves in polynomial time the NP-complete PIC. This leads to a contradiction, unless $P = NP$.

We show that the cost of a solution of the MCT instance is either 1, or at least $r+1$. If, on all the machines, for each pair (i, j) of tasks on the same machine we have $\alpha_{i,j} = 0$, then the maximum cost of a task is 1 (its own size, 1, times $\alpha_{i,i} = 1$). Otherwise, there is a machine with two tasks of types i and j with $\alpha_{i,j} = r$. The maximum cost is thus larger than or equal to the cost of task i, which is at least $1 \times \alpha_{i,i} + 1 \times \alpha_{i,j} = 1 + r$.

We show that the solution for the instance of PIC is "yes" if and only if there is a solution of cost 1 for the corresponding instance of MCT. Assume first that there is a solution for PIC: the vertices of G can be partitioned into $k \leq K$ disjoint sets $V_1, V_2, \ldots, V_k$ such that, for $1 \leq i \leq k$, the subgraph induced by V_i is a complete graph. For each $i \in \{1, \ldots, k\}$, we assign to machine M_i the tasks corresponding to the vertices of V_i. Since all the tasks on the same machine correspond to a clique in G, their coefficients $\alpha_{i,j}$ are all 0 (when $i \neq j$). The only cost of a task i is its own size times $\alpha_{i,i}$, that is 1. Thus, the cost of the optimal solution of the instance of MCT is 1.

Likewise, assume that there is a solution of cost 1 for the instance of MCT. Since the maximum cost of a task in the instance of MCT is 1, all the values $\alpha_{i,j}$ between the tasks on the same machine are 0 (for $i \neq j$), and thus that the corresponding vertices form a clique in G. Therefore, it is possible to partition the vertices of G into $m = K$ cliques: there is a "yes" solution for PIC. □

4 Approximation for fixed number of types

The inapproximability proof of the previous section means that we can develop constant-factor approximations only for MCT with a constant number of types (and constant coefficients). We show in this section two approximation algorithms: a PTAS and a fast greedy approximation algorithm called FillGreedy.

4.1 A PTAS

Our PTAS (Algorithm 1) has a similar structure to the PTAS proposed by Hochbaum and Shmoys for $P||C_{\max}$ [15]: it uses dichotomic search to find a minimal target maximum cost C (the target is the makespan in [15]). During the search, for a certain C, if the optimal cost is at most C, the algorithm returns a $(1+\varepsilon)$-approximate schedule; otherwise, the algorithm detects that no such schedule exists.

In order to build a schedule close to the optimum, the algorithm partitions the tasks into two sets: the long tasks and the small tasks. The long tasks are rounded down to the nearest multiple of some given number X. The small tasks of a same type are gathered together in some new long tasks of size X called containers (a more detailed description follows). This gives us a modified instance made of the rounded long tasks and the newly introduced container tasks. Since the number of different sizes in this instance is probably (much) smaller than in the original instance, these tasks are scheduled optimally using dynamic programming. Compared to the original PTAS of Hochbaum and Shmoys, the two main differences are the treatment of short tasks (which, in our algorithm, are not simply greedily scheduled, but are packed into containers), and the sizes of the long tasks. Our PTAS works even if $\alpha_{i,i} \neq 1$. The algorithm uses the following constant parameters: a number C denoting the requested maximum cost; an integer k; and $\gamma = T\alpha_{\max}\Big(2 + 1/(\min \alpha_{i,i})\Big)$ (we assume that T and $\alpha_{i,j}$ are constants). Given C, the algorithm either returns a schedule of cost at most $C(1+1/k)$, or proves that a schedule of cost at most C does not exist.

Algorithm 1: A PTAS for MCT with constant T and α

```
$J' = \emptyset$;
for $j \in J, p_j \geq C/(\gamma k)$ do // round down long tasks
    $p_{j'} = p_j - (p_j \mod C/(\gamma k)^2)$ ;
    $J' = J' \cup \{j'\}$ ;
for $t \in T$ do // glue short tasks to containers
    $W_s^{(t)} = \sum_{j \in J^{(t)}, p_j < C/(\gamma k)} p_j$ ; // load of small tasks of type t ;
    while $W_s^{(t)} > 0$ do
        $p_{j''} = \min(C/(\gamma k), W_s^{(t)})$ ;
        $J' = J' \cup \{j''\}$ ; // $j''$ is a new container ;
        $W_s^{(t)} = W_s^{(t)} - p_{j''}$ ;
for $t \in T$ do remove from $J'$ $m$ containers of type $t$ ;
$\sigma'^{*}$ = partition of $J'$ by solving (by dynamic programming)
$OPT(n_1'^{(1)}, \ldots, n_{(\gamma k)^2}'^{(1)}, \ldots, n_1'^{(T)}, \ldots, n_{(\gamma k)^2}'^{(T)}) =$
$1 + \min_{s_1^{(1)}, \ldots, s_{(\gamma k)^2}^{(T)} \in \mathscr{C}} OPT(n_1'^{(1)} - s_1^{(1)}, \ldots, n_{(\gamma k)^2}'^{(T)} - s_{(\gamma k)^2}^{(T)})$;
if $\sigma'^{*}$ requires more than $m$ machines then return $\emptyset$;
$\sigma = \sigma'^{*}$ ;
for $k$=1 to $m$ do // add removed containers
    for $k$=1 to $T$ do $\sigma[k] = \sigma[k] \cup \{C/(\gamma k)\}$;
for $k$=1 to $m$ do // replace containers by small tasks
    for $t \in T$ do
        $i$ = number of type $t$ containers in $\sigma[k]$ ;
        replace $i$ containers by tasks of total load $W$,
        $iC/(\gamma k) \leq W \leq (i+1)C/(\gamma k)$;
Replace in $\sigma$ rounded long tasks with original long tasks;
```

The algorithm starts by constructing an instance I' for which the optimal cost is a lower bound of the optimal cost for the original instance I. The algorithm partitions tasks into two sets: long tasks of size at least $C/(\gamma k)$, and the remaining short tasks (tasks of size smaller than $C/(\gamma k)$). Long tasks are rounded down to the nearest multiple of $X = C/(\gamma k)^2$. Short tasks of a single type are "glued" into *container tasks* of sizes $C/(\gamma k)$, except the last container task which might be shorter (of size $W_s^{(t)}$ mod $(C/(\gamma k)$, where $W_s^{(t)}$ is the load of short tasks of type t: $W_s^{(t)} = \sum_{j \in J^{(t)}, p_j < C/(\gamma k)} p_j$). Then, the algorithm reduces the load in container tasks by removing m containers (the shortest one and $m-1$ others) of each type. Note that if the total load of short tasks of type t is smaller than $mC/(\gamma k)$, there are less than m containers, and they are all removed in this step; later, when reconstructing schedule, the algorithm adds the number of containers that have been removed. We omit this detail from Algorithm 1 to make the code more readable. The number of tasks in I' is smaller than or equal to the number of tasks in I, and the total load in I' is smaller than or equal to the total load in I (the number of tasks and the load does not change only if all the tasks are long and their sizes are multiples of $C/(\gamma k)^2$).

The algorithm then schedules the tasks of I' using dynamic programming. For a given configuration $n_1^{'(1)}, \ldots, n_{(\gamma k)^2}^{'(1)}, \ldots, n_1^{'(T)}, \ldots, n_{(\gamma k)^2}^{'(T)}$, where $n_i^{'(t)}$ is the number of tasks in I' of type t and size $iC/(\gamma k)^2$, OPT denotes the minimal number of machines needed to schedule the configuration with cost smaller than C. Let $s_i^{(t)}$ be a number of tasks of type t and of size $iC/(\gamma k)^2$. To find OPT, the dynamic programming approach checks all the possible configurations $\mathscr{C}$ of task sizes for a single machine $s_1^{(1)}, \ldots, s_{(\gamma k)^2}^{(T)}$ that result in cost smaller than or equal t to C, i.e.: $s_1^{(1)}, \ldots, s_{(\gamma k)^2}^{(T)} \in \mathscr{C} \Leftrightarrow \forall t$ such that $\sum_i s_i^{(t)} > 0 : \sum_{t'} \sum_{i=1}^{(\gamma k)^2} \alpha_{t',t} s_i^{(t')} iC/(\gamma k)^2 \leq C$. If OPT is larger than m, the algorithm ends. Otherwise, the returned schedule $\sigma^{'*}$ forms a scaffold to build a schedule σ for the original instance I. First, the algorithm adds a container for each type on each machine (this container was removed before the dynamic programming). Then, the algorithm replaces containers by actual short tasks. Assume that $\sigma^{'*}$ scheduled $i-1$ containers of type t on machine m; the previous step added at most one container. The algorithm replaces i containers of a total load $iC/(\gamma k)$ by scheduling unscheduled short tasks of type t with a total load of at least $iC/(\gamma k)$ and at most $(i+1)C/(\gamma k)$ (which is always possible as a short task is shorter than $C/(\gamma k)$). Finally, the algorithm replaces long tasks that were rounded down by the original long tasks.

Proposition 3. *The PTAS returns a solution to* MCT *if and only if there is a solution of* MCT *of cost at most C. Moreover, if such a solution exists, the cost of the solution returned by the PTAS is at most $C(1+1/k)$.*

Proof. Assume first that there is an optimal schedule σ^* of instance I using m machines and having cost at most C. Consider a schedule σ' for I' constructed according to σ^*. Each long task in σ' is placed on the same machine as in σ^*. If σ^* executes on a machine a total load $W_s^{*(t)}$ of small tasks of type t, this load is replaced by $\lfloor W_s^{*(t)}/(C/(\gamma k) \rfloor$ containers, each of size $C/(\gamma k)$. σ' is a valid schedule for I' as it schedules all tasks in I'. Moreover, the cost of σ' is at most C, as the load of each type on each machine is not higher than the corresponding load in σ^*. As the dynamic programming used in PTAS analyses all possible schedules, it will return a schedule $\sigma^{'*}$ using at most the same number of machines as in σ^*.

Assume now that the dynamic programming returns a schedule $\sigma^{'*}$ of cost at most C. By adding a single container on each machine and each type, the cost increases by at most $T\alpha_{\max}C/(\gamma k)$. By replacing the containers by small tasks, the load of each type is increased by at most $C/(\gamma k)$, thus the cost is increased by at most $T\alpha_{\max}C/(\gamma k)$. Finally, by replacing the rounded-down long tasks by the tasks of the original sizes, the size of each long task is increased by at most $C/(\gamma k)^2$. As the cost of $\sigma^{'*}$ was at most C, and a long task is of size at least $C/(\gamma k)$, there are at most $\gamma k/\alpha_{i,i}$ tasks of each type i (as the cost of each type on itself has to be smaller than C). There are thus at most $T\gamma k/(\min \alpha_{i,i})$ long tasks in total. The total increase of cost due to long tasks (and the maximal influence between types) is thus at most $\left(T\gamma k/(\min \alpha_{i,i})\right)\alpha_{\max}\left(C/(\gamma k)^2\right)$. Consequently, the cost of σ is bounded by $C(\sigma) \leq C + \left(CT\alpha_{\max}/(\gamma k)\right)\left(2+1/(\min \alpha_{i,i})\right) = C(1+1/k)$ (as $\gamma = T\alpha_{\max}\left(2+1/(\min \alpha_{i,i})\right)$). □

Proposition 4. *The PTAS runs in polynomial time $O(n^{T(\gamma k)^2})$.*

Proof. T is a constant. We assume that $n \geq m$ as otherwise the optimal solution is trivial: each task is allocated to a different machine. Thus, the runtime of all loops in Algorithm 1 is bounded by $O(n)$. We upper-bound the cost of the dynamic programming by computing the number of valid entries of the $(n_1^{'(1)}, \ldots, n_{(\gamma k)^2}^{'(1)}, \ldots, n_1^{'(T)}, \ldots, n_{(\gamma k)^2}^{'(T)})$ vector. This vector has $(T(\gamma k)^2)$ dimensions, and $n_i^{'(t)} \leq n$ for each pair (i,t). Thus there are at most $n^{T(\gamma k)^2}$ distinct vectors. For each of these vectors, the algorithm must check at most as many entries as there are possible single machine configurations. As there are at most $\gamma k/\alpha_{i,i}$ tasks of type i on a single machine, there are at most $\left(1+\gamma k/(\min \alpha_{i,i})\right)^{T(\gamma k)^2}$ such configurations to check. As γ, $\alpha_{i,j}$, T and k are constant, the number of configurations to check is a constant; thus, the complexity of the dynamic programming algorithm is dominated by the number of valid entries to check, $O(n^{T(\gamma k)^2})$. □

4.2 A greedy list-scheduling approximation

We propose FILLGREEDY, a greedy $\frac{2Tm}{m-T}$-approximate algorithm for MCT with a constant number of types. FILLGREEDY groups tasks by *clusters*. All the tasks of the same type are in the same cluster. Two tasks of different types i and j are in the same cluster only if their types are compatible ($\alpha_{i,j} \leq 1$ and $\alpha_{j,i} \leq 1$). While minimizing the number of clusters is NP-hard (by an immediate reduction from PARTITION INTO CLIQUES), any heuristics can be used, as the approximation ratio does not depend on the number of clusters.

Clusters are processed one by one. At least one machine is dedicated to each cluster. We assume that m, the number of machines, is smaller than or equal to the number of clusters K. This is a realistic assumption since $K \leq T$, and in a data center, T should be much smaller than m. Let $L = (\sum p_i)/(m-T)$, and let $L_{\max} = \max\{2L, L + p_{\max}\}$. The algorithm considers the tasks cluster by cluster. It puts tasks from the current cluster on a machine until the load of the machine reaches $L_{\max}$. Then, it "opens" a new machine and puts tasks on this new machine again until the load reaches $L_{\max}$. This continues until all the tasks of the current cluster have been assigned to a machine. In other words, a task of the currently considered cluster is scheduled on a new machine if and only if the current load on the current machine plus the size of the task is larger than L_{max}. When all tasks of a cluster have been scheduled, the algorithm schedules in the same way and starting on a new machine, the tasks of the following cluster. We demonstrate (Proposition 5) that the number of machines is sufficient to schedule the tasks without having to open a $(m+1)$-st machine. However, in practice, in order to minimize the maximum cost, rather than fixing the maximum machine load to $L_{\max}$, we perform a dichotomic search over $[1, L_{\max}]$ to find the smallest possible threshold leading to a feasible allocation. Once the clusters are created, the complexity of FILLGREEDY with dichotomic search over $[1, L_{\max}]$ is $O(n \log(L_{\max}))$ (there are $\log(L_{\max})$ steps, and assigning a machine to each of the n tasks is done in $O(1)$).

Proposition 5. *Algorithm* FILLGREEDY *is a* $\frac{2Tm}{m-T}$*-approximate algorithm for* MCT.

Proof. We first show that the allocation is feasible, i.e. the algorithm uses at most m machines. Let m_{used} be the number of machines to which at least one task is allocated. Among these m_{used} machines, at most K have load smaller than L. Indeed, for each cluster the algorithm allocates tasks to a machine beyond L (as $L_{\max} \geq L + p_{max}$), unless there are no remaining task. Thus, for each cluster, only the load of the last opened machine can be smaller than L. Thus, the load allocated on these m_{used} machines is at least $(m_{used} - K)L = (m_{used} - K)\frac{W}{m-T}$. Since the total load is W, we have $(m_{used} - K)\frac{W}{m-T} \leq W$. Thus $\frac{m_{used}-K}{m-T} \leq 1$, and so $m_{used} - K \leq m - T$. Since $K \leq T$, we have $m_{used} \leq m$. Thus, the allocation returned by FILLGREEDY is feasible.

We now show that the cost is $\frac{2Km}{m-T}$-approximate. We consider an instance I of MCT. Let $\mathscr{O}$ be an optimal solution of I for MCT, and let OPT be the maximum cost of a task in $\mathscr{O}$. Since, for each type i, $\alpha_{i,i} = 1$, we have $OPT \geq p_{max}$. Let $L_{max}(\mathscr{O})$ be the maximum load of a machine in $\mathscr{O}$. Let us consider that this load is achieved on machine i. We have $L_{max}(\mathscr{O}) \geq \frac{W}{m}$ (by the surface argument). Since there are at most T types on machine i, there is at least one type which has a load of at least $\frac{L_{max}(\mathscr{O})}{T}$ on machine i. The cost of a task of this type on machine i is thus at least $\frac{L_{max}(\mathscr{O})}{T}$, and therefore $OPT \geq \frac{L_{max}(\mathscr{O})}{T} \geq \frac{W}{Tm}$.

Let $\mathscr{S}$ be the solution returned by FILLGREEDY for instance I. Let $C(\mathscr{S})$ be the maximum cost of a task in $\mathscr{S}$. Let $L_{max}(\mathscr{S})$ be the maximum load of a machine in $\mathscr{S}$. Since two tasks i and j are allocated to the same machine only if they belong to the same cluster, i.e. only if $\alpha_{t_i,t_j} \leq 1$, the cost of each task is at most equal to $L_{max}(\mathscr{S})$, and thus $C(\mathscr{S}) \leq L_{max}(\mathscr{S})$. Moreover, by construction, we have $L_{max}(\mathscr{S}) \leq \max\{2L, L + p_{max}\}$. We consider the two following cases:

- $\max\{L, p_{max}\} = p_{max}$:
$C(\mathscr{S}) \leq L_{max} \leq L + p_{max} = \frac{W}{m-T} + p_{max} = \left(\frac{Tm}{m-T}\right)\frac{W}{Tm} + p_{max}$. Since we know that $OPT \geq p_{max}$ and $OPT \geq \frac{W}{Tm}$, we have $C(\mathscr{S}) \leq (\frac{Tm}{m-T} + 1)OPT < \frac{2Tm}{m-T}OPT$.
- $\max\{L, p_{max}\} = L$:
$C(\mathscr{S}) \leq L_{max} \leq 2L = \frac{2W}{m-T} = 2\left(\frac{Tm}{m-T}\right)\frac{W}{Tm} \leq \frac{2Tm}{m-T}OPT$ because $OPT \geq \frac{W}{Tm}$. □

5 Heuristics for any number of types

In this section, we propose other algorithms for MCT. We later show (in Section 6) that these algorithms are fast approximations when there are two types. Each of these algorithms uses as a sub-procedure an algorithm $\mathscr{A}$ solving $P||C_{\max}$. This procedure can be for example the LPT (Longest Processing Times first) list algorithm, or an approximation scheme for $P||C_{\max}$. The input of $\mathscr{A}$ is the number of machines and the set of tasks of our problem: we thus do not consider types and we convert p_i, the size of task i in our problem, to a simple duration of the task. In Section 6, we will prove approximation ratios of our algorithms for MCT as a function of the approximation ratio of $\mathscr{A}$.

- SCHEDMIXED applies $\mathscr{A}$ on all tasks and all machines. Let σ be the schedule constructed by $\mathscr{A}$ on m machines with tasks J. SCHEDMIXED($\mathscr{A}$) returns the partition P of the tasks corresponding to the allocation in σ (tasks on M_i in P are the tasks on M_i in σ). SCHEDMIXED thus corresponds to the baseline approach — ignoring the types and allocating tasks only by their size.
- SCHEDJUXTAPOSE first applies algorithm $\mathscr{A}$ on each of the T types separately: for each type t, $\mathscr{A}$ schedules $J^{(t)}$, the set of tasks of type t, on the m machines. Then, SCHEDJUXTAPOSE juxtaposes (joins) the T obtained schedules. Let σ_t be the schedule obtained by applying $\mathscr{A}$ on tasks $J^{(t)}$ of type t on m machines. SCHEDJUXTAPOSE merges the schedules by reversing the order of the machines for every other type: the tasks on machine M_i in the returned schedule are the tasks allocated to M_i in σ_{2k+1} and the tasks allocated to M_{m-i+1} in σ_{2k}. Note that such reordering does not change the approximation ratio proved in Section 6 — compared to the schedule where each machine M_i contains the tasks of M_i of every schedule σ_t. However, when $\mathscr{A} = LPT$, in practice, this allows to decrease the makespan, and thus the maximum cost.
- BESTSCHEDULE($\mathscr{A}$) returns the partition with the lowest cost among the results of SCHEDJUXTAPOSE($\mathscr{A}$) and SCHEDMIXED($\mathscr{A}$).
- GREEDYDEDICATED($\mathscr{B}$) takes in input, besides the set of tasks and the number of machines, types grouped into K clusters such that each pair of types in each cluster is compatible (as in Section 4.2). Machines are partitioned into K sets — a set for each cluster — and two tasks of different clusters are not assigned to the same machine. For each cluster i, and for each possible number of machines $m_i \in \{1,\ldots,m-K+1\}$, GREEDYDEDICATED runs an algorithm $\mathscr{B}$ (one of SCHEDMIXED, SCHEDJUXTAPOSE or BESTSCHEDULE), on the tasks of cluster i, on m_i machines. GREEDYDEDICATED returns an allocation of the minimal cost over all legal partitions of machines onto K clusters by exhaustive search over vectors $[m_i] : \sum_{i=1}^{K} m_i = m$. For example, when there are two types and $\alpha > 1$, each cluster contains one type. GREEDYDEDICATED executes for each $m_1 \in \{1,\ldots,m-1\}$, algorithm $\mathscr{B}$ twice: first, for tasks $J^{(1)}$ of type 1 allocated on the first m_1 machines; second, for tasks $J^{(2)}$ of type 2 allocated on the remaining $m - m_1$ machines. Out of these $(m-1)$ allocations, GREEDYDEDICATED returns the allocation with the smallest cost.

Let $C_{\mathscr{A}}$ be the complexity of Algorithm $\mathscr{A}$. Algorithm SCHEDMIXED is in $O(C_{\mathscr{A}})$; SCHEDJUXTAPOSE and BESTSCHEDULE are in $O(TC_{\mathscr{A}})$; GREEDYDEDICATED is in $O(Km^K C_{\mathscr{A}})$.

6 Special case: two types

To show how the coefficient α shapes the optimal allocation, we study in this section a series of special cases with only two types ($T = 2$) and a symmetric coefficient $\alpha = \alpha_{t',t} = \alpha_{t,t'}$. This special case also enables us to prove approximation ratios of the algorithms introduced in the previous section. For one of the sub-cases we propose a slight refinement of FILLGREEDY called GREEDYFOR2TYPES that achieves a better approximation ratio.

We distinguish three cases based on α: *compatible* types when $\alpha \leq 1$; *incompatible* when $1 < \alpha < 2$; and *clashing* when $\alpha \geq 2$. These boundary values have the following motivation.

To illustrate the difference between *compatible* and *incompatible* types, we consider divisible loads instances. The definition follows the one used in scheduling: if $W^{(i)}$ is the load of the tasks of type $i \in \{1,2\}$, this load can be assigned in any way to the machines (as if it was composed of a huge number of tiny tasks each of size ε).

When $\alpha < 1$, the types are *compatible*. A task from a different type results in a smaller cost than a task from the same type. Thus, the optimal allocation of I evenly shares all machines between the two types: each machine executes a load $W^{(i)}/m$ of type i.

When $\alpha = 1$, any allocation that has load $(W^{(1)} + W^{(2)})/m$ on each machine is optimal (the impact of a task of type 1 or 2 on another task is the same).

When $\alpha > 1$, a task from a different type results in a larger cost than a task from the same type. Thus, in *divisible load instances*, there is at most one machine that is shared between the two types. If there were more than one shared machine, we could reduce the cost by exchanging tasks between the machines. Note that for standard (non-divisible load) instances, there might be more than one shared machine.

To distinguish the cases of *incompatible* and *clashing* types, we now focus on non divisible load instances. When $\alpha < 2$, there can be up to m machines which have to be shared in an optimal solution, as in the following instance. There are m tasks of each type: type 1 has only long tasks (of length p); type 2 has only short tasks (of length ε). In an allocation with m shared machines, the maximum cost is $\alpha p + \varepsilon$. In an allocation with less than m shared machines, at least one machine executes two tasks of type 1, so the maximum cost is at least $2p$. Thus, if $\alpha < 2$, and if ε is sufficiently small, a schedule with m shared machines has a lower cost. In contrast if $\alpha \geq 2$, we prove in Proposition 11 that there is always at most a single shared machine.

6.1 Compatible types ($\alpha \leq 1$)

Proposition 6. *Let $\mathscr{A}$ be a $O(X)$, $(1+\varepsilon)$-approximate algorithm for $P||C_{max}$. Algorithm* SCHEDJUXTAPOSE*($\mathscr{A}$) is a $O(X)$, $(1+\varepsilon)(1+\alpha)$-approximate algorithm for* MCT *for $T = 2$ and $\alpha \leq 1$.*

Proof. Let I be an instance of MCT. Let P be the partition returned by algorithm SCHEDJUXTAPOSE($\mathscr{A}$) for instance I. Let $Cost(P)$ be the cost of P. Let P^* be an optimal solution of instance I for MCT, and let OPT be the cost of P^*. Let C_1^* (resp. C_2^*) be the makespan of an optimal schedule of I_1 (resp. I_2) for problem $(P||C_{max})$. Let C_1 (resp. C_2) be the makespan of schedule S_1 (resp. S_2). We have: $C_1 \leq (1+\varepsilon)C_1^*$ and $C_2 \leq (1+\varepsilon)C_2^*$ since $\mathscr{A}$ is a $(1+\varepsilon)$-approximate algorithm for problem $(P||C_{max})$. Moreover, we have $C_1^* \leq OPT$. Indeed, in P^* all the tasks of I_1 are partitioned into at most m subsets (machines): the maximum load of tasks of type 1 on a same machine in P^* is at least C_1^*. Since the cost of a task of type 1 is at least the load of the tasks of type 1 on the same machine (because $\alpha_{1,1} = 1$), we have $OPT \geq C_1^*$. Likewise, we have $OPT \geq C_2^*$. Let us assume without loss of generality that $C_1 \leq C_2$. The cost of P is smaller than or equal to $C_1 + \alpha C_2$, since the cost of a task of type 1 is the load of the tasks of type 1 on the same machine (at most C_1) plus $\alpha \leq 1$ times the load of the tasks of type 2 on the same machine (at most C_2). Thus $Cost(P) \leq C_1 + \alpha C_2 \leq (1+\varepsilon)(C_1^* + \alpha C_2^*) \leq (1+\varepsilon)(1+\alpha)OPT$. □

Proposition 7. *Let $\mathscr{A}$ be a $(1+\varepsilon)$-approximate algorithm for $P||C_{max}$.* SCHEDMIXED*($\mathscr{A}$) is a $\frac{2(1+\varepsilon)}{1+\alpha}$-approximate algorithm for* MCT *for $T = 2$ and $\alpha \leq 1$.*

Proof. Let I be an instance of MCT. Let P be the partition returned by algorithm SCHEDMIXED($\mathscr{A}$) for instance I. Let $Cost(P)$ be the cost of P.

Let C_{max}^* be the makespan of an optimal solution of problem $(P||C_{max})$ on instance I. Let C_{max} be the makespan of the schedule returned by $\mathscr{A}$ on instance I. Since $\mathscr{A}$ is a $(1+\varepsilon)$-approximate algorithm for problem $(P||C_{max})$, we have $C_{max} \leq (1+\varepsilon)C_{max}^*$. Moreover, $Cost(P) \leq C_{max}$ since the cost of each task is equal to the load of the tasks of the same type on the same machine times α times the load of the tasks of the other type on the same machine, and $\alpha \leq 1$.

Let P^* be an optimal solution of instance I for MCT, and let OPT be the cost of P^*. Let M_i be the most loaded machine in P^* and let $C_{max}(P^*)$ be the makespan of P^* (i.e. $C_{max}(P^*)$ is equal to the sum of the sizes of the tasks on M_i in P^*). Let L_1 (resp. L_2) be the load of the tasks of type 1 (resp. type 2) on M_i in P^*. Without loss of generality, assume that $L_1 \geq L_2$. The cost of the tasks of type 1 on M_i is $L_1 + \alpha L_2$. Thus, $OPT \geq L_1 + \alpha L_2 = L_1 + \alpha(C_{max}(P^*) - L_1) \geq \frac{C_{max}(P^*)}{2} + \alpha\frac{C_{max}(P^*)}{2} = (\frac{1+\alpha}{2})C_{max}(P^*)$. The last inequality holds because $L_1 \geq \frac{C_{max}(P^*)}{2}$ and $\alpha \leq 1$.

Since $Cost(P) \leq C_{max} \leq (1+\varepsilon)C_{max}^*$ and $OPT \geq (\frac{1+\alpha}{2})C_{max}(P^*) \geq (\frac{1+\alpha}{2})C_{max}^*$, we have $Cost(P) \leq \frac{2(1+\varepsilon)}{1+\alpha}OPT$. □

SCHEDJUXTAPOSE has lowest approximation for α close to 0, while SCHEDMIXED has lowest approximation for α close to 1.

Proposition 8. *Let $\mathscr{A}$ be a $(1+\varepsilon)$-approximate algorithm for problem $(P||C_{max})$, which runs in $O(X)$. Algorithm* BESTSCHEDULE*($\mathscr{A}$) is a $O(X)$, $\sqrt{2}(1+\varepsilon)$-approximate algorithm for* MCT *for $T = 2$ and $\alpha \leq 1$.*

Proof. The approximation ratio of SCHEDJUXTAPOSE($\mathscr{A}$) is $(1+\varepsilon)(1+\alpha)$ (Proposition 6). The approximation ratio of SCHEDMIXED($\mathscr{A}$) is $\frac{2(1+\varepsilon)}{1+\alpha}$ (Proposition 7). Thus, the approximation ratio of algorithm BESTSCHEDULE($\mathscr{A}$) is $\min\{(1+\varepsilon)(1+\alpha), \frac{2(1+\varepsilon)}{1+\alpha}\}$. The maximum is achieved when $(1+\varepsilon)(1+\alpha) = \frac{2(1+\varepsilon)}{1+\alpha} \Rightarrow (1+\alpha) = \frac{2}{1+\alpha} \Rightarrow (1+\alpha)^2 = 2 \Rightarrow \alpha^2 + 2\alpha - 1 = 0$ Since $\alpha \geq 0$, this means that $\alpha = \frac{-2+\sqrt{8}}{2} = \sqrt{2} - 1$. The maximum approximation ratio is obtained for $\alpha = \sqrt{2} - 1$. It is thus $(1+\varepsilon)(1+\alpha) = \sqrt{2}(1+\varepsilon)$. Therefore, BESTSCHEDULE($\mathscr{A}$) is $\sqrt{2}(1+\varepsilon)$-approximate. Furthermore, if $\mathscr{A}$ runs in $O(X)$, then SCHEDJUXTAPOSE($\mathscr{A}$) also runs in $O(X)$ since it runs twice algorithm $\mathscr{A}$, and algorithm SCHEDMIXED($\mathscr{A}$) also runs in $O(X)$ since it simply runs once algorithm $\mathscr{A}$. Therefore, algorithm BESTSCHEDULE($\mathscr{A}$), which runs once SCHEDJUXTAPOSE($\mathscr{A}$) and once SCHEDMIXED($\mathscr{A}$) also runs in $O(X)$. □

Corollary 1. *Let LPT be the algorithm which greedily schedules n tasks in decreasing order of their lengths on parallel machines. Algorithm* BESTSCHEDULE*(LPT) runs in $O(n \log n)$ and has an approximation ratio of $\frac{4\sqrt{2}}{3} < 1.89$ for* MCT.

6.2 Incompatible types ($1 < \alpha < 2$)

In this section, we show that SCHEDMIXED is a $\alpha(1+\varepsilon)$ approximation when $1 < \alpha \leq 2$. We also introduce GREEDYFOR2TYPES, a 2-approximate greedy algorithm running in $O(n)$, and which refines FILLGREEDY (whose approximation ratio is in this case $\frac{4m}{m-2}$, i.e., 4 for a large number of machines.)

Proposition 9. *Let $\mathscr{A}$ be a $(1+\varepsilon)$-approximate algorithm for problem $(P||C_{\max})$, which runs in $O(X)$. Algorithm* SCHEDMIXED *returns an $\alpha(1+\varepsilon)$-approximate solution for* MCT *in $O(X)$, for $T = 2$ and $1 < \alpha \leq 2$.*

Proof. The proof is similar to the one of Proposition 7. Let us consider an instance I of MCT. This instance can be considered as an instance of problem $(P||C_{\max})$, by considering m machines and tasks of lengths $p_1 \ldots, p_n$. Let $C_{\max}$ be the makespan of the schedule returned by $\mathscr{A}$ on I and let $C_{\max}^*$ be the optimal makespan of a schedule of I for problem $(P||C_{\max})$. We have $C_{\max} \leq (1+$

$\varepsilon)C^*_{\max}$ since $\mathscr{A}$ is a $(1+\varepsilon)$-approximate algorithm for problem $(P||C_{\max})$. Let us now consider the schedule returned by $\mathscr{A}$ as a solution for MCT (the assignment of the tasks to the machines is the one done in the schedule returned by $\mathscr{A}$). The maximum load of a machine in this solution is, by definition, $C_{\max}$. Since $\alpha_{i,i} < \alpha$, the cost of a task on a machine is smaller than α times the load of its machine, and thus smaller than $\alpha C_{\max}$. Likewise, since the maximum load of any solution of MCT on I is at least $C^*_{\max}$, we have $OPT \geq C^*_{\max}$. Hence, the maximum cost of a task in the solution returned by $\mathscr{A}$ is at most $\alpha C_{\max} \leq \alpha(1+\varepsilon)C^*_{\max} \leq \alpha(1+\varepsilon)OPT$. □

We now define a fast greedy algorithm for MCT called GreedyFor2Types. Like FillGreedy, GreedyFor2Types first assigns all the tasks of type 1, and then all the tasks of type 2. It fills the machines until a threshold and opens a new machine for the first task of the second type. Tasks gradually fill machines (as in FillGreedy): a task (other than the first task of the second type) is assigned to the current machine if the resulting total load on that machine is at most $L = W/m + \max\{W/m, p_{\max}\}$; otherwise, a new machine is opened. The difference between the two algorithms is that, with GreedyFor2Types, one machine can be shared between types. Indeed, if the algorithm tries to open machine $m+1$, then all the remaining tasks are assigned to the last machine on which a task of type 1 is assigned.

Proposition 10. *Algorithm* GreedyFor2Types *is a $O(n)$, 2-approximate algorithm for* MCT *for $T = 2$ and $1 < \alpha \leq 2$.*

Proof. Let M_x be the last machine on which a task of type 1 is assigned. By construction, M_x is the only machine which might have tasks of both types. On all the machines except M_x, the load is at most $L = W/m + \max\{W/m, p_{\max}\}$. Let OPT be the cost of an optimal solution for MCT. Since $\alpha \geq 1$, $OPT \geq p_{\max}$ and $OPT \geq W/m$. Thus, the cost of each task executing on machine different than M_x is at most $L \leq 2OPT$.
We now show that the cost of a task assigned to M_x is also at most $2OPT$. If there are only tasks of type 1 on M_x, then the load of this machine is at most L (otherwise, the total allocated load of type 1 would be greater than W), thus the cost of the tasks on M_x is at most $L \leq 2OPT$. If there are tasks of type 2 on M_x, on all other machines the load is larger than W/m. Thus the load on M_x is smaller than $W - (m-1)W/m = W/m$. Therefore, the cost of a task on M_x is smaller than $\alpha W/m \leq 2OPT$ since $\alpha \leq 2$. Hence, the solution returned by GreedyFor2Types is 2-approximate. □

6.3 Clashing types ($\alpha \geq 2$)

For large coefficients, we show that an optimal solution uses at most one shared machine. We then use this result to show that an algorithm that uses no shared machine is a $(1+\frac{1}{1+\alpha})(1+\varepsilon)$ approximation.

Proposition 11. *If $\alpha \geq 2$, there is an optimal solution that uses at most one shared machine.*

Proof. The proof is by contradiction. Let $\alpha \geq 2$. Assume that there is an instance in which all optimal solutions have at least two shared machines. Let us consider, for this instance, an optimal solution with a minimal number of shared machines (this number is thus at least two). Let M_k and $M_{k'}$ be two shared machines in this solution.

Let us assume that machine M_k executes load of $a = W_k^{(1)}$ of type 1 and $b = W_k^{(2)}$ of type 2. Machine $M_{k'}$ executes load of $a' = W_{k'}^{(1)}$ of type 1 and $b' = W_{k'}^{(2)}$ of type 2. Without loss of generality, we label types such that type 1 has higher load ($a + a' \geq b + b'$) and machines such that machine M_k is allocated most of type 1 load ($a \geq a'$). If the maximal cost of a task on these two machines is larger than or equal to $\max(a+a', b+b') = a+a'$, then we put tasks of type 1 on M_k and tasks of type 2 on $M_{k'}$ and the maximum cost of these tasks will be $\max(a+a', b+b') = a+a'$. As we have not increased the maximum cost, and decreased by two the number of shared machines, we contradict the assumption that we started with a solution having a minimal number of shared machines.

Let us now assume that the maximum cost of the tasks of M_k and $M_{k'}$ is smaller than $\max(a+a', b+b') = a+a'$. We consider two cases. In the first case, $a \geq b$. On M_k, the maximum cost is then the cost of tasks of type 2. It is equal to $b + \alpha a$. We have assumed that the maximum cost on M_k and $M_{k'}$ is strictly smaller than $a+a'$, so $b + \alpha a < a + a'$. As $b > 0$, $\alpha a < a + a'$. As $a \geq a'$, $\alpha a < 2a$, which leads to a contradiction since $\alpha \geq 2$.

In the second case, $a < b$. Thus, on M_k, the maximum cost is the cost of tasks of type 1. It is $a + \alpha b$, and $a + \alpha b > a + \alpha a > \alpha a$. We have assumed that the maximum cost on M_k and $M_{k'}$ is smaller than $a + a'$, thus $\alpha a < a + a' \leq 2a$, which, as in the previous case, leads to a contradiction since $\alpha \geq 2$. □

We now consider algorithm GreedyDedicated, defined in Section 5.

Proposition 12. *Given a $(1+\varepsilon)$-approximate algorithm $\mathscr{A}$ for $P||C_{\max}$ which runs in $O(X)$,* GreedyDedicated *is a $O(mX)$, $(1+\frac{1}{1+\alpha})(1+\varepsilon)$-approximate algorithm for* MCT *for $T = 2$ and $\alpha \geq 2$.*

Proof. For $\alpha \geq 2$, there is an optimal solution σ^* with at most one shared machine (Proposition 11). Let OPT be the cost of σ^* for MCT. We consider two cases. If there is no shared machine in σ^*, then we define $\sigma' = \sigma^*$. If there is a shared machine M_i in σ^*, assume that this machine executes load $W_i^{(1)}$ of type 1 and $W_i^{(2)}$ of type 2. Thus, $W_i^{(1)} + W_i^{(2)} \leq \frac{2OPT}{1+\alpha}$, and therefore we have $\min\{W_i^{(1)}, W_i^{(2)}\} \leq \frac{OPT}{1+\alpha}$. We now construct from σ^* a schedule σ' that will not use a shared machine. The smaller out of $W_i^{(1)}$ and $W_i^{(2)}$ is moved to the first machine dedicated for its type. The cost of σ' is at most $OPT(1+\frac{1}{1+\alpha})$ and it is equal to the makespan on some machine (there is no shared machine anymore).

Let us consider now the allocation returned by GreedyDedicated and let us denote its makespan by $C_{\max}$. As $\alpha_{t,t} = 1$ and no machine is shared, the maximum cost is equal to the makespan, $C_{\max}$. Let $C_{max}^{*(noshared)}$ be the minimal makespan of a schedule in which there is no shared machine. Assume that this optimal schedule uses m_1^* machines for type 1. GreedyDedicated tests all $m_1 \in \{1, \ldots m\}$. For each m_1 (including m_1^*) it executes a $(1+\varepsilon)$-approximate algorithm $\mathscr{A}$. Thus, the makespan $C_{\max}$ returned by GreedyDedicated is at most $(1+\varepsilon)C_{\max}^{*(no\ shared)}$.

Since no machine is shared in σ', $C_{\max} \leq (1+\varepsilon)C_{\max}^{*(no\ shared)} \leq (1+\varepsilon)C_{\max}(\sigma') \leq (1+\varepsilon)OPT(1+\frac{1}{1+\alpha})$. □

7 Experiments

7.1 Method

7.1.1 Data

We use the cluster trace from Google [30], the standard dataset for datacenter/cloud resource management research. The trace describes all tasks running during a month on one of the Google clusters. For each task, the trace reports in its task record table, among other data, the task's CPU, memory and disk IO usage averaged over a 5-minute long period. This trace is certainly not ideal for our needs: the trace reports the usage of raw resources (CPU, memory, network, disk), and not the load of applications. However, to our best knowledge, there are no publicly-available traces describing loads and performance of applications (in contrast to raw resources).

We generate a random sample of 10,000 task records (see [27] for more information on the sample). Each task record corresponds to a task in our model. To generate loads and types, we use data on the mean CPU utilization and the assigned memory. We normalize CPU and memory utilization to their respective maximums. We remove 45% of task records that reported less than 0.005 in both normalized CPU and normalized memory usage.

To assign one of T types to a task, we analyze the ratio ρ of the weighted CPU to the weighted memory usage. For $T = 2$, a task with $\rho \leq 1$ is of type 1 (memory-intensive), and a task of $\rho > 1$ is of type 2 (CPU-intensive). This partitions the dataset in almost equal halves. For $T = 3$, we pick thresholds $\log(\rho_1) = -0.66$ and $\log(\rho_2) = 0.66$. We chose these thresholds from the histogram of the distribution of ρ — they correspond to values of ρ for which the number of tasks significantly diminishes. Type 1 is memory-intensive (10%), type 3 is CPU-intensive (11%) and type 2 is a mixed CPU-memory task. An alternative would be to use the disk IO measurements also reported in the trace. However, in only 3% of tasks the normalized disk IO dominates both CPU and memory: this would result in 3-type instances having a very small number of tasks of the 3rd type, and these instances would thus be very close to 2-type instances. For $T = 4$ we use thresholds of $\log(\rho_1) = -0.66$, then $\rho_2 = 1$, and $\log(\rho_3) = 0.66$. Type 1 is memory-intensive (10%), type 2 is memory-CPU (40%), type 3 is CPU-memory (39%), and type 4 is CPU-intensive (11%).

To assign load to a task, we take the maximum from the weighted CPU and weighted memory, multiply this maximum by 100 and round it to the nearest integer.

To generate an instance of n tasks belonging to T types (with $T \in \{2,3,4\}$), we take a random sample of n tasks from the dataset. Thus, the proportions of types in the generated instance are similar to the dataset. However, a random sample might have less than T types: if it is the case, we remove a task from the most common type in the instance and add a task of a missing type.

We generate the coefficients α in four different ways. In all instances coefficients are normalized ($\alpha_{t,t} = 1$) and generated from a uniform distribution from the following ranges:

- *compatible*: coefficients are smaller than 1;
- *incompatible*: coefficients are between 1 and 2;
- *clashing*: coefficients are between 1 and 4;
- *mixed*: there are 2 incompatible clusters (see Section 4.2). In instances with 3 types, t_1 and t_2 are mutually compatible and incompatible with t_3. Similarly, in instances with 4 types, types t_1 and t_2 are compatible; types t_3 and t_4 are compatible; but both t_1 and t_2 are incompatible with both t_3 and t_4.

Note that the way we set the coefficients in non-compatible scenarios does not necessarily correspond to the way we partition the trace into types. We continue with these discretionary values as, first, we want to test our algorithms for variety of settings; and, second, we are not aware of any better dataset.

We generate an instance by choosing one of the four different ways of setting the coefficients, a number of types $T \in \{2,3,4\}$, and a category of instance – small or large. In *small* instances, the number of tasks is $n \in \{10,20,50,100,200,500,1000\}$ and the number of machines is $m \in \{2,3,5,10\}$ (we generate all possibilities). In *large* instances, the number of tasks is $n \in \{200,500,1000,2000,5000\}$ and the number of machines is $m \in \{20,50,100\}$ (again, we generate all possibilities). Finally, we discard unfeasible combinations: for *incompatible* and *clashing* coefficients, instances in which the number of types is higher than the number of machines; for *mixed* instances, instances in which the number of types is smaller than 3. For each feasible combination, we generate 30 instances. Overall, we generate 12930 feasible instances.

7.1.2 Algorithms

We study all the proposed algorithms except the PTAS. We briefly recall each algorithm below:

- SCHEDMIXED (denoted by *mix* in plots, Sections 5 and 6.1): the algorithm allocates tasks to machines without using the information on types; it is thus a baseline for the type-based approaches.
- FILLGREEDY, (*fill* in plots, Section 4.2): the algorithm allocates compatible types to common machines loading each machine until a threshold. We use binary search to optimize the threshold up to which each machine is loaded. We also sort tasks in each cluster by decreasing lengths (which makes our algorithm analogous to the last-fit decreasing bin packing algorithms).
- SCHEDJUXTAPOSE (*jux* in plots, Section 5): the algorithm allocates each type separately on all machines and then juxtaposes, or joins, the allocations. When juxtaposing schedules of different types, we reverse the order of machines for every other type (as in LPT with a small number of tasks, the machines with smallest indices have the highest load).
- BESTSCHEDULE (*best* in plots, Section 5: the algorithm internally runs SCHEDMIXED and SCHEDJUXTAPOSE and then returns the allocation with the lowest cost among the two).
- GREEDYDEDICATED (*ded* in plots, Sections 5 and 6.3): the algorithm separates types into compatible clusters; incompatible types do not share machines; the algorithm runs one of SCHEDMIXED, SCHEDJUXTAPOSE or BESTSCHEDULE to allocate tasks within each cluster.
- GREEDYFOR2TYPES (*g2* in plots), (Section 6.2): a modification of FILLGREEDY that shares up to one machine between types.
- FILLREBALANCE (*fill-r* in plots), a further modification of GREEDYFOR2TYPES: as in GREEDYFOR2TYPES (and as in FILLGREEDY) the algorithm starts by assigning tasks to machines by types: each type opens a new machine and machines are filled up to a certain treshold (as in FILLGREEDY, we optimize this treshold with binary search). When the algorithm runs out of new machines, it tries to assign each of the remaining tasks to one of the m machines (it might mix types in this step). To test whether a task fits on a machine, the algorithm computes the cost (in contrast to GREEDYFOR2TYPES which computes load).

We use LPT as $\mathscr{A}$, the underlying scheduling algorithm for the single type problem ($P||C_{\max}$). LPT orders tasks by decreasing sizes; tasks are processed sequentially and each task is assigned to a machine with the smallest total load (note that we consider here the load and not the cost).

We run FILLGREEDY on all instances; SCHEDJUXTAPOSE, SCHEDMIXED and BESTSCHEDULE on *compatible* instances; SCHEDMIXED, GREEDYFOR2TYPES, FILLREBALANCE and GREEDYDEDICATED on *incompatible* and *clashing* instances. On *mixed* instances, we run SCHEDMIXED (as in Section 6.2), GREEDYFOR2TYPES and FILLREBALANCE. We also run GreedyDedicated between the two clusters. In this case the algorithm used inside the clusters (algorithm $\mathscr{A}$) is either SchedJuxtapose (denoted by $d - jux$ in plots), SchedMixed ($d - mix$ in plots) or BestSchedule ($d - best$ in plots).

7.1.3 Scoring

For meaningful comparisons of algorithms' results across instances having vastly different loads, we compute the relative scores by weighting the maximum cost returned by an algorithm on an instance to a lower bound. As a lower bound, we use a maximum from $p_{\max}$ and a solution of a following quadratically-constrained program with binary variables (QCP). $p_{\max}$, the maximum size of the task, is a lower bound of the cost for instances in which $\alpha_{t,t} = 1$ for each type t. Indeed, each task has to be allocated to some machine; and the lowest cost the task imposes on its type is when the task is alone (placing any other task with $\alpha_{t',t} > 0$ increases type's t cost on this machine). The QCP is the following:

$$\min c, \text{such that} \tag{2}$$

$$\forall k \in [1,m], \forall \tau : u_{\tau,k} \sum_t x_{t,k} W^{(t)} \alpha_{t,\tau} \leq c \tag{3}$$

$$\forall t : \sum_k x_{t,k} = 1, \tag{4}$$

$$\forall k \in [1,m], \forall t : 0 \leq x_{t,k} \leq 1, \tag{5}$$

$$\forall k,t : u_{t,k} \in \{0,1\}, \tag{6}$$

$$\forall k,t : x_{t,k} \leq u_{t,k}. \tag{7}$$

This program computes the cost c in a relaxed version of our problem. We allocate fractions of loads $W^{(t)}$ rather than individual, discrete tasks. A decision variable $x_{t,k}$ specifies a fraction of type's t total load $W^{(t)}$ allocated on machine k. To calculate type's t cost (Eq. 3), we take into account the load on a machine k iff t places some load on k. We use an auxiliary binary variable $u_{t,k}$ to model such usage ($u_{t,k} = 1$ iff $x_{t,k} > 0$, Eq. 7). We use the Gurobi solver [14] to solve this optimization problem. As the formulation uses binary variables $u_{t,k}$, it is significantly more difficult to solve than a standard linear program. If the solver does not finish in 15 minutes (which happened in 8% of the considered instances), we interrupt it and take the best lower bound on the optimum found so far.

7.2 Results

Figure 1 presents the normalized cost by instance size and type. All the results discussed below are statistically-significant (two sided paired t-test, p-values smaller than 0.0001). On average and across all instance types, the type-oblivious SchedMixed results in higher costs than type-aware methods.

Incompatible and clashing coefficients (Fig. 1c, 1d, 1g, 1h) test the behavior of the algorithms when sharing a machine between types might be more costly than dedicating machines to types. On these instances SchedMixed is the sole algorithm that consistently shares a machine between types. Our results clearly show that its costs are the highest, thus they underline the motivation for type-aware approaches. FillGreedy and GreedyDedicated allocate a single type to a machine on incompatible and clashing coefficients (the algorithms operate on clusters of mutually-compatible types, but in these instances all types are mutually incompatible, thus the clusters contain exactly a single type). FillRebalance slightly improves FillGreedy. In instances with small number of machines, sharing a machine might lead to a better allocation. FillRebalance reduces the mean cost in small, *clashing* instances from 1.09 to 1.07; and in small, *incompatible* instances from 1.15 to 1.08. In large instances, the reduction is less than 1% — when there are more machines, sharing one or two has a smaller impact. Except for small, *incompatible* instances, GreedyDedicated produces allocations with the lowest cost among the three: its mean costs are 1.20 for small, incompatible instances and 1.02 for large, incompatible instances. The results of GreedyDedicated on *clashing* instances (Fig. 1d and 1h) are almost identical to the results on *incompatible* instances (Fig. 1c and 1g) — the minor differences are caused by random generation of other parameters of the instance.

Instances with compatible coefficients (Fig. 1a and 1e) highlight the consequences of different ways machines can be shared between types (thus the differences between FillGreedy, SchedMixed and SchedJux). FillGreedy has the highest average costs among these algorithms. On the average, SchedMixed, the algorithm allocating jobs as if there were no types, produces schedules with lower cost than SchedJux, the algorithm allocating each type separately and only then joining the allocations (SchedMixed means are 1.10 for small instances and 1.18 for large). However, BestSchedule, choosing for each instance the best out of SchedMixed and SchedJux has even lower costs (1.04 for small, 1.07 for large), demonstrating that it is occasionally better to use the type-aware SchedJux.

Finally, mixed coefficients (Fig. 1b and 1f) test both sharing (between compatible types) and dedicating machines (to incompatible clusters). As expected, the algorithm combining the best-performing algorithms from the instances discussed above — GreedyDedicated using BestSchedule inside clusters — dominates other algorithms with means 1.14 for small instances and 1.03 for large ones. We clearly see the advantage of using type-aware algorithms, as SchedMixed (mixing incompatible clusters) has a significantly higher mean score (1.41 for small instances, 1.62 for large).

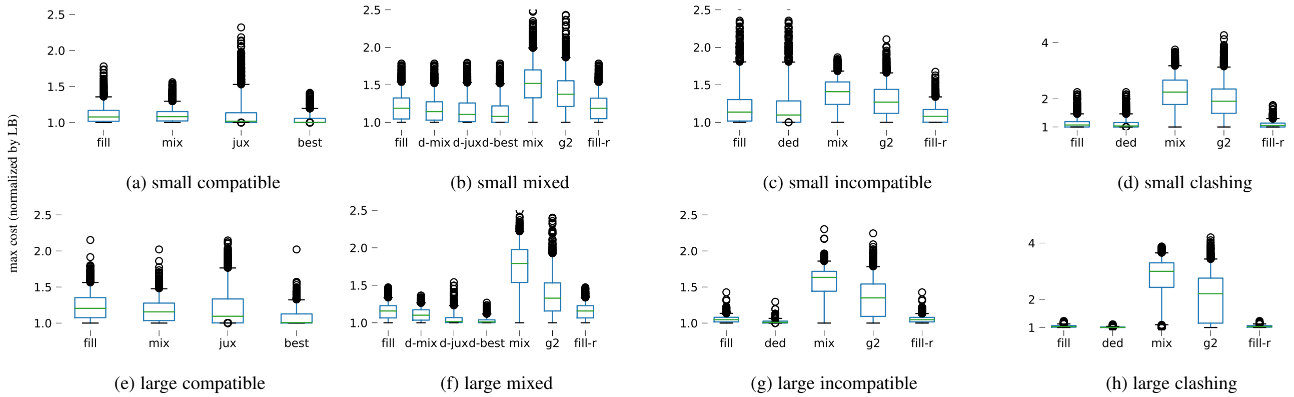
8 Related Work

We introduced the side-effects performance model [26], [27], where we studied a utilitarian (min-sum) objective. We proved that the problem is NP-hard and we showed a dominance property (for each type, there is an order of the machines such that the tasks are assigned by decreasing sizes to the machines). This allowed us to give an exact polynomial time algorithm when there is a single type. For the general case, we proposed two algorithms, which are exponential in the number of types and either the number of machines or the number of admissible tasks' sizes.

Alternative models of data center resource management.

Cloud resource management is a thriving research area. In the brief description below, we do not aim to provide a complete bibliography for this rapidly expanding field, especially as there is a recent survey [28]; we rather contrast our approach with papers representative of various approaches.

Many colocation performance models are too complex for combinatorial results [18], [23], [24]. Schedulers rely on heuristic approaches with no formal performance guarantees [4], [6], [17], [34]. In *bin-packing* approaches (e.g., [31], [33], [16]), tasks are modeled as items to be packed into bins (machines) of known capacity [7]. To model heterogeneity, bin packing is extended to vector packing: item's size is a vector with dimensions corresponding to requirements on individual resources (CPU,



(a) small compatible (b) small mixed (c) small incompatible (d) small clashing

(e) large compatible (f) large mixed (g) large incompatible (h) large clashing

Fig. 1. The maximum cost of the solutions returned by various heuristics normalized by the lower bound. All instances. In boxplots the middle line represents the median, the box spans between the 25th and the 75th percentile, the whiskers span between the 5th and the 95th percentile, and the circles show the remaining points (outliers).

memory, disk or network bandwidth) [32]. Garefalakis et al. [10] additionally considers placement constraints (e.g., tasks' affinity and anti-affinity). Mann [25] adds joint optimization of VMs to PMs and application components to VMs. Alternatively, if tasks have unit-size requirements, simpler representations can be used, such as maximum weighted matching [2]. Bin packing and related approaches assume that machines' capacities are crisp and that, as long as machines are not overloaded, any allocation is equally good for tasks. In our model, machines' capacities are not crisp—instead, tasks' performance gradually decreases with increased load.

Flow-based approaches (e.g. [12]) encode affinities and constraints in a flow graph and then solve min cost max flow problem to determine the optimal (or approximate) placement for many tasks simultaneously. Gog et al. [12] report excellent placement latency (an aspect we have not addressed in this paper—we only demonstrated acceptable time complexity of our proposed algorithms). However, like bin-packing, flow-based approaches also do not consider performance degradation (beyond encoding task affinities and anti-affinities).

Statistical approaches. Bobroff et al. [3] use 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. Cortez et al. [8] use machine learning to predict VM resource consumption for more efficient oversubscription.

Di et al. [9] 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, optimality conditions are formulated. When the resources are insufficient, fair scheduling policies are proposed.

Analysis of effects of colocation. Studies showing performance degeneration when colocating data center tasks include [19], [21], [36], [37]. Xu et al. [37] forms models of cloud tasks performance as a function of their resource consumption and colocation and proposes an optimization algorithm based on a metaheuristic. Podzimek et al. [29] analyze the performance of colocated CPU-intensive benchmarks. Kim et al. [20] measure performance of several colocated HPC applications. For each pair of colocated applications, they measure the runtime normalized by the runtime the application has without a co-runner. Our $\alpha_{t,t'}$ coefficients are similar in spirit (and could possibly be set by) their interference/affinity metrics. Additionally [20] give a greedy allocation heuristic, but they don't study its worst-case performance nor the complexity of the problem.

Related scheduling models. Considering tasks with types and (in)compatibilities between types is close in spirit to scheduling tasks with setup times [1]. Each task has a type from a set $\mathscr{T}$ of types, and for each possible couple of types $(i, j) \in \mathscr{T}^2$, there is a setup time $s_{i,j}$ if a task of type i is scheduled just before a task of type j. The difference between this problem and ours is that setup times consider tasks scheduled sequentially over the time—the setup time between two tasks t and t' delays task t' and all the tasks scheduled after t'. In contrast in our case tasks are scheduled concurrently, thus a task influences all the tasks scheduled on the same machine.

9 Conclusion

We considered a problem of allocation of tasks to machines in the side-effects performance model. Performance of a task depends on the load of other tasks colocated on the same machine. We use a linear performance function: the influence of tasks of type t' on a task of type t is their total load times a coefficient $\alpha_{t',t}$ which describes how compatible types t' and t are. We minimize the maximal cost. We prove that this NP-hard problem is hard to approximate if there are many types. However, handling a limited number of types is feasible: we show a PTAS and a fast approximation algorithm, as well as a series of heuristics that are approximation algorithms for two types. We simulate allocations resulting from algorithms on instances derived from one of Google clusters. Our simulations show that algorithms taking into account types lead to significantly lower costs than non-type algorithms.

Our results show a possible way to adapt to data centers the large body of work in scheduling, which development has been often inspired by advances in HPC platforms. We deliberately chose to study a fundamental problem, a minimal extension to $P||C_{\max}$. We envision that more realistic variants of data center resource management problem, taking into account release dates, non-clairvoyance or on-line, can be taken into account similarly as they are considered in classic scheduling.

Acknowledgements We thank Pierre Fouilhoux for hepful discussions which allowed us to improve the lower bound (Section 7.1.3). We also thank anonymous reviewers for an idea that lead to FillRebalance. This research has been partly supported by a Polish National Science Center grant Opus (UMO-2017/25/B/ST6/00116), and a Polonium grant (joint programme of the French Ministry of Foreign Affairs, the Ministry of Science and Higher Education and the Polish Ministry of Science and Higher Education).

References

[1] Allahverdi, A., Ng, C., Cheng, T., Kovalyov, M.Y.: A survey of scheduling problems with setup times or costs. European Journal of Operational Research 187(3), 985 – 1032 (2008)

[2] Beaumont, O., Eyraud-Dubois, L., Thraves Caro, C., Rejeb, H.: Heterogeneous resource allocation under degree constraints. IEEE TPDS 24(5), 926–937 (2013)

[3] Bobroff, N., Kochut, A., Beaty, K.: Dynamic placement of virtual machines for managing SLA violations. In: IFIP/IEEE International Symposium on Integrated Network Management (IM), Proc. pp. 119–128. IEEE (2007)

[4] Bu, X., Rao, J., Xu, C.z.: Interference and locality-aware task scheduling for mapreduce applications in virtual clusters. In: HPDC, Proc. pp. 227–238. ACM (2013)

[5] Cheng, Y., Anwar, A., Duan, X.: Analyzing alibaba's co-located datacenter workloads. In: 2018 IEEE International Conference on Big Data (Big Data). pp. 292–297. IEEE (2018)

[6] Chiang, R.C., Huang, H.H.: Tracon: interference-aware scheduling for data-intensive applications in virtualized environments. In: SC Proc. p. 47. ACM (2011)

[7] Coffman Jr, E.G., Garey, M.R., Johnson, D.S.: Approximation algorithms for bin packing: A survey. In: Hochbaum, D. (ed.) Approximation algorithms for NP-hard problems. pp. 46–93. PWS (1996)

[8] Cortez, E., Bonde, A., Muzio, A., Russinovich, M., Fontoura, M., Bianchini, R.: Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In: Proceedings of the 26th Symposium on Operating Systems Principles. pp. 153–167. ACM (2017)

[9] Di, S., Kondo, D., Wang, C.: Optimization of composite cloud service processing with virtual machines. IEEE Transactions on Computers 64, 1755–1768 (2015)

[10] Garefalakis, P., Karanasos, K., Pietzuch, P.R., Suresh, A., Rao, S.: Medea: scheduling of long running applications in shared production clusters. In: EuroSys. pp. 4–1 (2018)

[11] Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness (1979)

[12] Gog, I., Schwarzkopf, M., Gleave, A., Watson, R.N., Hand, S.: Firmament: Fast, centralized cluster scheduling at scale. In: Usenix, Proc. (2016)

[13] Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAP 17(2), 416–429 (1969)

[14] Gurobi Optimization, L.: Gurobi optimizer reference manual (2019), http://www.gurobi.com

[15] Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems theoretical and practical results. Journal of the ACM 34(1), 144–162 (1987)

[16] Janus, P., Rzadca, K.: SLO-aware colocation of data center tasks based on instantaneous processor requirements. In: ACM SoCC Proc. pp. 256–268. ACM (2017)

[17] Jersak, L.C., Ferreto, T.: Performance-aware server consolidation with adjustable interference levels. In: SAC Proc. pp. 420–425 (2016)

[18] Jin, X., Zhang, F., Wang, L., Hu, S., Zhou, B., Liu, Z.: Joint optimization of operational cost and performance interference in cloud data centers. ToCC (2015)

[19] Kambadur, M., Moseley, T., Hank, R., Kim, M.A.: Measuring interference between live datacenter applications. In: SC, Proc. p. 51. IEEE (2012)

[20] Kim, S., Hwang, E., Yoo, T.K., Kim, J.S., Hwang, S., Choi, Y.R.: Platform and co-runner affinities for many-task applications in distributed computing platforms. In: IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Proc. pp. 667–676. IEEE CS (2015)

[21] Koh, Y., Knauerhase, R., Brett, P., Bowman, M., Wen, Z., Pu, C.: An analysis of performance interference effects in virtual environments. In: ISPASS, Proc. pp. 200–209. IEEE (2007)

[22] Koutsoupias, E., Papadimitriou, C.: Worst-case equilibria. In: STACS, Proc. pp. 404–413. Springer (1999)

[23] Kundu, S., Rangaswami, R., Dutta, K., Zhao, M.: Application performance modeling in a virtualized environment. In: HPCA. pp. 1–10. IEEE (2010)

[24] Kundu, S., Rangaswami, R., Gulati, A., Zhao, M., Dutta, K.: Modeling virtualized applications using machine learning techniques. In: SIGPLAN Not. vol. 47, pp. 3–14. ACM (2012)

[25] Mann, Z.Á.: Resource optimization across the cloud stack. IEEE TPDS 29(1), 169–182 (2018)

[26] Pascual, F., Rzadca, K.: Partition with side effects. In: HiPC 2015, Procs. (2015)

[27] Pascual, F., Rzadca, K.: Colocating tasks in data centers using a side-effects performance model. EJOR 268(2), 450–462 (2018)

[28] Pietri, I., Sakellariou, R.: Mapping virtual machines onto physical machines in cloud computing: A survey. CSUR 49(3), 49 (2016)

[29] Podzimek, A., Bulej, L., Chen, L.Y., Binder, W., Tuma, P.: Analyzing the impact of CPU pinning and partial CPU loads on performance and energy efficiency. In: IEEE/ACM CCGrid), Proc. pp. 1–10. IEEE CS (2015)

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

[31] Song, W., Xiao, Z., Chen, Q., Luo, H.: Adaptive resource provisioning for the cloud using online bin packing. IEEE ToC 63(11), 2647–2660 (2014)

[32] Stillwell, M., Vivien, F., Casanova, H.: Virtual machine resource allocation for service hosting on heterogeneous distributed platforms. In: IPDPS, Proc. pp. 786–797. IEEE (2012)

[33] Tang, X., Li, Y., Ren, R., Cai, W.: On first fit bin packing for online cloud server allocation. In: IPDPS, Proc. pp. 323–332 (2016)

[34] Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., Wilkes, J.: Large-scale cluster management at Google with Borg. In: EuroSys Proc. p. 18. ACM (2015)

[35] Vöcking, B.: Selfish load balancing. In: Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.V. (eds.) Algorithmic Game Theory, pp. 517–542. Cambridge (2007)

[36] Xu, Y., Musgrave, Z., Noble, B., Bailey, M.: Bobtail: Avoiding long tails in the cloud. In: USENIX NSDI, Proc. pp. 329–341 (2013)

[37] Zhao, H., Wang, J., Liu, F., Wang, Q., Zhang, W., Zheng, Q.: Power-aware and performance-guaranteed virtual machine placement in the cloud. IEEE TPDS 29(6), 1385–1400 (2018)

Fanny Pascual received a PhD in computer science from Evry University, France, in 2006. Between 2006 and 2007 she did a post-doc at INRIA, in Grenoble, France. Since 2007, she has been an associate professor at Sorbonne Université, in Paris, France. Her research interests include design and analysis of algorithms for combinatorial optimization problems, and especially for problems involving different users. She with a special interest on resource management and scheduling problems.

Krzysztof Rzadca is an associate professor in the Institute of Informatics, University of Warsaw, Poland. He graduated with a MSc in software engineering from Warsaw University of Technology, Poland in 2004, and a PhD in computer science in 2008 jointly from Institut National Polytechnique de Grenoble (INPG), France and from Polish-Japanese Institute of Information Technology, Poland. He was French government fellowship recipient during his PhD studies. Between 2008 and 2010, he worked as a research fellow in Nanyang Technological University (NTU), Singapore. He was awarded grants from the Polish National Science Center, the Foundation for Polish Science and a faculty research award from Google. His research focuses on resource management and scheduling in large-scale distributed systems.