Optimizing egalitarian performance in the side-effects model of colocation for data center resource management

Fanny Pascual[1] and Krzysztof Rzadca[2]

[1] Sorbonne Universités, UPMC, LIP6, CNRS, UMR 7606, Paris, France
Email: fanny.pascual@lip6.fr
[2] Institute of Informatics, University of Warsaw, Poland
Email: krz@mimuw.edu.pl

Abstract. In data centers, up to dozens of tasks are colocated on a single physical machine. Machines are used more efficiently, but tasks' performance deteriorates, as colocated tasks compete for shared resources. As tasks are heterogeneous, the resulting performance dependencies are complex. In our previous work [18] 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: maximizing the worst-off performance. 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 and 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. Algorithms aware of jobs' types lead to better performance compared to algorithms solving $P||C_{\max}$.
The notion of type enables us to model degeneration of performance caused by colocation using standard combinatorial optimization methods. Types add a layer of additional complexity. However, our results — approximation algorithms and good average-case performance — show that types can be handled efficiently.

Keywords: cloud computing; scheduling; heterogeneity; co-tenancy; complexity

1 Introduction

The back-bone of cloud computing, the modern data center redefines how industry and academia use computers. Resource management in data centers significantly differs from scheduling jobs on a typical HPC supercomputer. First, the workload is much more varied [22]: data centers act as a physical infrastructure providing virtual machines, or higher-level services, such as memory-cached databases or network-intensive servers; in contrast, there are relatively few HPC-like computationally-intensive batch jobs (later, we will use a generic term *task* for all these categories). Consequently, a task usually does not saturate the resources of a single node [12]. Tasks' loads vastly differ:

in a published trace [22], tasks' average CPU loads span more than 4 orders of magnitude. In contrast to HPC scheduling in which jobs rarely share a node, heterogeneity in both the type and the amount of needed resources makes it reasonable to allocate multiple tasks to the same physical machine.

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 [12–14, 21, 27] that the performance of colocated tasks drops. Suspects include difficulties in sharing the CPU cache or the memory bandwidth. The resource manager should thus colocate tasks that are compatible, i.e., that use different kinds of resources — hence, it should optimize tasks' performance. This, however, requires a performance model.

Our side-effects model [18] bridges the gap between colocation in datacenters and the theoretical scheduling, bulk of which has been developed for non-shared machines.

Rather than trying to predict tasks' performance from OS-level metrics, we abstract by characterizing a task by two characteristics: *type* (e.g.: a database, or a computationally-intensive job) and *load* 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 loads of tasks of the i-th type located 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 function can be inferred by a monitoring module [13, 21, 27] matching task's reported performance (such as the 95th percentile response time) with observed or reported loads.

In this paper, we consider optimization of the worst-off performance (analogous to makespan in classic multiprocessor scheduling problem, $P||C_{\max}$ [8]). We use a linear performance function: on each machine, the influence a type t' has on type t performance is a product of the load of type t' and a coefficient $\alpha_{t',t}$. The coefficient $\alpha_{t',t}$ describes how compatible t' load is with t performance (the coefficient is similar to interference/affinity metrics proposed in [13, 21]). Low values ($0 \leq \alpha_{t',t} < 1$) correspond with 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. High values ($\alpha_{t'',t} > 1$) denote types competing for resources.

The contribution of this paper is 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 and hard to approximate when the number of types T is not constant (Section 3). We then show how to cope with that added complexity. We propose (2) a PTAS for a constant T and a constant α (Section 4.1); and (3) a fast greedy approximation algorithm (Section 4.2). (4) We also propose natural greedy heuristics (Section 5) (in the accompanying technical report [19] we show they are approximations for $T = 2$). (5) We also test our algorithm by simulation on a trace derived from one of Google clusters (Section 6).

2 Side-Effects of Colocating Tasks: A Model

We study a min-max (egalitarian) performance criteria for our side-effects performance model (introduced in [18], where we studied a utilitarian objective, min-sum). We con-

sider a system that allocates n tasks $J = \{1,\dots,n\}$ to m identical machines $\mathscr{M} = \{M_1,\dots,M_m\}$. Each task i has a known size $p_i \in \mathbb{N}$ (i.e., clairvoyance, a common assumption in scheduling; the sizes can be estimated by previous instances 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 tasks are known (off-line) and ready to be scheduled (released at time 0). We take these assumptions to derive results on the basic model before tackling more complex ones. We denote by $p_{\max} = \max p_i$ the largest task and by W the total load, $W = \sum p_i$. 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 main contribution of this paper lies in analyzing side-effects of colocating tasks. 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 and may have different granularities: for instance, "a webserver" and "a database"; or "a read-intensive MySQL database"; or, as in [13], "an instance of Blast" . We assume that the type t_i is known (which again corresponds to the clairvoyance assumption in classic scheduling; typically a data center runs many instances of the same task, so task's type can be derived from the past). Let $\mathscr{T} = \{1,\dots,T\}$ be a set of T different types of tasks. Each task i has type $t_i \in \mathscr{T}$. For each type $t \in \mathscr{T}$, we denote by $J^{(t)}$ the tasks which are of type t; and 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. Task's i cost c_i depends on to the *total load* of tasks j colocated on the same machine $M_{P,i}$, but different types have different impacts:$c_i = \sum_j$ on machine $_{M_{P,i}}\, p_j.\alpha_{t_j,t_i}$,The cost function also takes into account the task i itself, as well as 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 in function of their colocation and their sizes (a data center runs many instances of similar services [13,21,27]). We consider the linear cost function as it generalizes, by adding coefficients $\alpha_{t,t'}$, the fundamental scheduling problem $P||C_{\max}$ [8] (if $\forall (t,t') \in \mathscr{T}^2 : \alpha_{t,t'} = 1$, our problem reduces to $P||C_{\max}$). Assuming linearity is a common approach when constructing models in operational research or statistics (e.g.

linear regressions). Likewise, in selfish load balancing games [15], it is assumed that the cost of each task is the total load of the machine (but 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 by MSE (MINMAXCOST WITH SIDE EFFECTS) the problem of finding a partition P^* minimizing the maximum cost $C(P) = \max_i c_i$, with c_i defined by the linear cost function. The partition P^* minimizes the worst performance a task experiences in the system, thus corresponds to the egalitarian fairness.

3 Complexity and hardness of MSE for T not fixed

MSE is NP-hard as it generalizes an NP-hard problem $P||C_{\max}$ when there is only one type. Our main result is that a polynomially-solvable variant of mupltiprocessor 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-complete $P||C_{\max}$.

Proposition 1. *The decision version of* MSE *is NP-complete, even if all the tasks have unit size, and even if* $m = 2$.

Proof. (Sketch) Reduction from PARTITION [7]. Given a set $S = \{a_i\}$ of n positive integers summing to $2B$, we build an instance of MSE with n tasks, each of size 1 and each of a different type. For a task i, we set its coefficients $\forall j : \alpha_{i,j} = a_i$. Partition of S into two sets each with sum B exists if and only if there exists an allocation P with maximal cost B: cost of each task j allocated to a machine k is equal to $c_j = \sum_{i:M_{P,i}=k} \alpha_{i,j} = \sum_{i \in S_k} a_i$.

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

Proof. Let $r > 1$. We show that if there is a r-approximate algorithm for MSE, the algorithm solves NP-complete PARTITION INTO CLIQUES, PIC [7]. In PIC, given a graph $G = (V,E)$ and a positive integer $K \leq |V|$, can the vertices of G 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? We assume that V are labeled from 1 to $|V|$.

Given an instance of PIC, we create K instances of MSE. Let $i \in \{1, \ldots, K\}$. The i-th instance of MSE is as follows: the number of machines is $m = i$; there are $n = |V|$ tasks, each of a different type (types are labeled from 1 to $|V|$). All the tasks are of size 1. For each type i, $\alpha_{i,i} = 1$. 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 a MSE 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 one of these MSE instances is 1. Therefore, an r-approximate algorithm for MSE will find a solution of cost 1 if it exists (when there is a solution of cost 1, an r-approximate algorithm has to return a solution of cost at most r, which is thus necessarily the optimal

solution since all the other solutions have a cost of at least $r+1$). Since $K \leq |V|$, if we assume that our r-approximate algorithm runs in polynomial time, then by using it K times we can solve in polynomial time PIC, which is an NP-complete problem. This leads to a contradiction, unless $P = NP$.

We show that the cost of a solution of each of the MSE instances 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 at least 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 (at least) one of the $|V|$ instances of MSE. 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. We take the k-th MSE instance. 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 ($i \neq j$). 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 k-th instance of MSE is 1.

Likewise, assume that there is a solution of cost 1 for (at least) one of the $|V|$ instances of MSE (wlog, for the k-th instance). Then there is a "yes" solution for PARTITION INTO CLIQUES: since the maximum cost for the instance of MSE is 1, it means that all the values $\alpha_{i,j}$ between tasks on the same machines are 0 (for $i \neq j$) and thus that corresponding vertices form a clique in G.

4 Approximation for fixed number of types

The inapproximability proof of the previous section means that we can develop constant-factor approximations only for MSE with a constant number of types (and constant coefficients). We show in this section two approximation algorithms. First, a PTAS running in time $O(n^{T(\gamma k)^2})$, and thus mostly of theoretical interest. Then we introduce a fast greedy approximation algorithm.

4.1 A PTAS

Our PTAS (Algorithm 1) has a similar structure to the PTAS for $P||C_{\max}$ [9]: the two main differences are the treatment of short tasks (which we pack into containers, and not simply greedy schedule) and the sizing of long tasks. Our PTAS works even if $\alpha_{i,i} \neq 1$, and $\alpha_{i,j} \neq \alpha_{j,i}$. The algorithm uses parameters: C, the requested maximum cost; k, an integer; and $\gamma = T\alpha_{\max}\left(2 + 1/(\min \alpha_{i,i})\right)$ (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.

The algorithm starts by constructing an instance I' which will form a lower bound for C of the original instance I. The algorithm partitions tasks into two sets: long tasks of size at least $C/(\gamma k)$; and short tasks. Long tasks are rounded down to the nearest multiple of $C/(\gamma k)^2$. Short tasks of a single type are "glued" into *container tasks*

Algorithm 1: A PTAS for MSE 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 ;
σ'^* = 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 σ'^* *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 σ rounded long tasks with original long 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 same number of containers that were removed. We omit this detail from Algorithm 1 to make the code more readable). The resulting instance I' has at most as many tasks and at most as high overall load as the original instance 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 lower-bound instance 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. To find OPT, the dynamic programming approach checks all possible configurations $\mathscr{C}$ of task sizes for a single machine $s_1^{(1)}, \ldots, s_{(\gamma k)^2}^{(T)}$ (where $s_i^{(t)}$ denotes the number of tasks) that result in cost smaller than 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 σ'^* 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 σ'^* 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* MSE *if and only if there is a solution of* MSE *of cost at most C. Moreover, if such a solution of cost C exists, the cost of the solution returned by the PTAS is at most* $C(1+1/k)$.

Proofs omitted due to space constraints are in the accompanying technical report [19].

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

4.2 A Greedy list-scheduling approximation

FILLGREEDY is a greedy $\frac{2Tm}{m-T}$-approximate algorithm for MSE with constant number of types. FILLGREEDY groups tasks by *clusters*. All the tasks of the same type are in the same cluster. Two tasks of type i and j are in the same cluster iff 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. Each cluster is allocated to at least one, dedicated machine. (We assume that m, the number of machines, is smaller than or equal to the number of clusters K; $K \leq T$, and in a data center T should be much smaller than m). The algorithm puts tasks from a cluster on a machine until machine load reaches $L_{\max} = \max\{2L, L+p_{\max}\}$ (where $L = (\sum p_i)/(m-T)$ is the average load), then opens the next machine. In practice, rather than fixing the maximum machine load to $L_{\max}$, we do a dichotomic search over $[1, L_{\max}]$ to find the smallest possible threshold leading to a feasible schedule. The complexity of FILLGREEDY with dichotomic search is $\mathcal{O}(T^2 n \log(L_{\max}))$.

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

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 tasks. 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 MSE. Let $\mathcal{O}$ be an optimal solution of I for MSE, and let OPT be the maximum cost of a task in $\mathcal{O}$. Since, for each type i, $\alpha_{i,i} = 1$, we have $OPT \geq p_{max}$. Let $L_{max}(\mathcal{O})$ be the maximum load of a machine in $\mathcal{O}$. 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 scheduled on 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:

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

– case 2: $\max\{L, p_{max}\} = L$. In this case, $C(\mathscr{S}) \leq L_{max} \leq 2L = \frac{2W}{m-T} \leq 2\left(\frac{Tm}{m-T}\right)\frac{W}{Tm} \leq \frac{2Tm}{m-T}OPT$ because $OPT \geq \frac{W}{Tm}$.

5 Heuristics

We propose a few other algorithms for MSE. These algorithms are fast approximations when $T = 2$ (see [19]). They all use as a subprocedure an algorithm $\mathscr{A}$ for $P||C_{\max}$, such as LPT (used in our experiments). $\mathscr{A}$ uses task's size p_i as task's length.

SCHEDMIXED uses $\mathscr{A}$ 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 equal to allocation in σ (tasks on M_i in P are the tasks on M_i in σ).

SCHEDJUXTAPOSE uses $\mathscr{A}$ on all machines for each type separately and then joins the schedules. Let σ_t be the schedule obtained by applying $\mathscr{A}$ on tasks $J^{(t)}$ of type t on m machines. SCHEDJUXTAPOSE merges schedules reversing the order of machines for every other type, i.e.: tasks on machine M_i are tasks allocated to M_i in σ_{2k+1} and to M_{m-i+1} in σ_{2k} (when $\mathscr{A} = LPT$ and with a small number of tasks, the machines with smallest indices have the highest load).

BESTSCHEDULE($\mathscr{A}$) returns the partition with the lowest cost among the results of SCHEDJUXTAPOSE($\mathscr{A}$) and SCHEDMIXED($\mathscr{A}$).

GREEDYDEDICATED($\mathscr{B}$) separates types into K clusters (as in Section 4.2). Clusters do not share machines. The algorithm runs a subprocedure $\mathscr{B}$ (SCHEDMIXED, SCHEDJUXTAPOSE or BESTSCHEDULE) to put tasks of k-th cluster onto m_k machines. GREEDYDEDICATED returns the allocation with the minimal cost over all possibilities of assigning $[m_k]$ to clusters (by exhaustive search over $[m_k] : \sum_{k \in K} m_k = m$).

The complexities of SCHEDMIXED, SCHEDJUXTAPOSE and BESTSCHEDULE are the same as $\mathscr{A}$; GREEDYDEDICATED, because of the exhaustive search, is exponential in T times the complexity of $\mathscr{A}$.

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 Experiments

We used the Google Cluster Trace [22], the standard dataset for datacenter/cloud resource management research, as an input data. The trace is certainly not ideal for our needs as it shows 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. Due to space constraints we describe the details of conversion in the accompanying technical report [19].

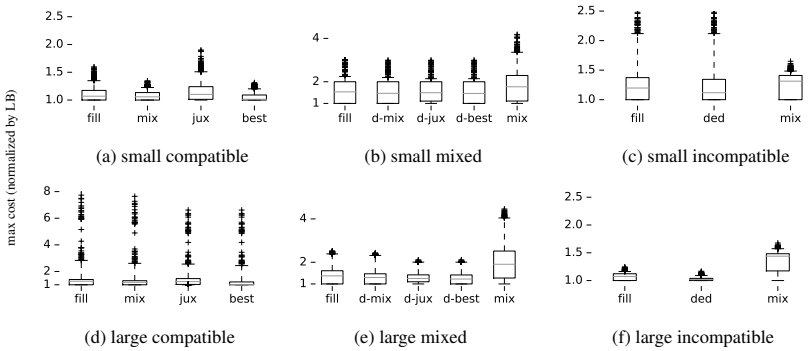
We generate a random sample of 10.000 task records. Each task record corresponds to a task in our model. To generate loads and types, we use data on the (normalized) mean CPU utilization and the assigned memory. Task's type is determined by the ratio of the the CPU to the memory usage. We generate the coefficients α in four different ways: *compatible*: smaller than 1; *incompatible*: between 1 and 2; *clashing*: at least 2; *mixed*: 2 incompatible clusters. There are $T \in \{2,3,4\}$ types. Instances have two sizes: in *small* ones, the there are $n \in \{10,20,50\}$ tasks and $m \in \{2,3,5,10\}$ machines; In *large* ones, there are $n \in \{200,500,1000\}$ tasks and $m \in \{20,50,100\}$ machines. For each combination we generate 30 instances; after discarding some unfeasible combinations (e.g., mixed, $T = 2$), we have 6390 instances.

We tested the following algorithms: FILLGREEDY (*fill* in plots); SCHEDMIXED (*mix*); SCHEDJUXTAPOSE (*jux*); BESTSCHEDULE (*best*); GREEDYDEDICATED with either SCHEDJUXTAPOSE ($d-jux$), SCHEDMIXED ($d-mix$) or BESTSCHEDULE ($d-best$). We omit some algorithms on some instances if they are clearly sub-optimal: GREEDYDEDICATED on compatible instances; and SCHEDJUXTAPOSE on all but compatible instances. On incompatible and clashing instances, all variants of GREEDYDEDICATED result in the same allocation—we denote the algorithm by *ded* in this case.

We compared the maximum cost returned by the algorithms to the lower bound and computed the relative performance. We used the following lower bounds. (1) $p_{\max}$, the maximum size of the task (as we assume $\alpha_{t,t} = 1$, the cost on the machine on which the longest task is allocated is at least $p_{\max}$). (2) For incompatible and clashing instances, the average load of a machine, W/m. (3) For compatible instances, a solution of a LP that optimizes the fraction of type t's load to be allocated to machine k. (4) For mixed instances, the same LP solved for each cluster on m machines (this lower bound effectively assumes mK available machines).

Figure 1 presents the normalized cost (scores on clashing and incompatible instances are exactly the same). We had two kinds of problems with the lower bound (details in [19]), both resulting in underestimation of the optimal solution and thus overestimation of the cost of our algorithms. First, the LP solver we used (python-scipy) often failed on large compatible instances: on 6% of $T = 3$ and 70% of $T = 4$ instances. Second, the LP lower bound underestimates the optimal cost of mixed instances. In such problematic instances, $p_{\max}$ was often used, which resulted in a lower bound that might significantly underestimate the cost of the optimum. To reduce the effect of such outliers, we discuss medians, rather than means, in the sequel.

Overall, all algorithms have similar performance and the performance is close to the lower bound except in mixed instances. On average, GREEDYDEDICATED, SCHEDMIXED and SCHEDJUX produce schedules with lower costs than FILL GREEDY (note that SCHEDMIXED and SCHEDJUX are used directly for compatible instances, and as



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

(d) large compatible (e) large mixed (f) large incompatible

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 asterisks show all the remaining points (outliers).

sub-procedures for GREEDYDEDICATED for the mixed instances); and BESTSCHEDULE optimizes even further. All the results below are statistically-significant (two sided paired t-test, p-values smaller than 0.0001).

Incompatible coefficients isolate the difference between FILL GREEDY, GREEDYDEDICATED, and SCHEDMIXED (as all clusters have a single type, neither FILL GREEDY nor GREEDYDEDICATED allocate different types onto a single machine). Results clearly show that sharing machines (SCHEDMIXED) leads to higher costs. GREEDYDEDICATED produces allocations with the lowest cost: its median costs are 1.02 for large instances and from 1.12 for small instances.

Compatible coefficients isolate the difference between FILL GREEDY, SCHEDMIXED and SCHEDJUX. On the average, SCHEDMIXED produces schedules with a lower cost than SCHEDJUX (medians are 1.06 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 (1.01 for both small and large), demonstrating the need to occasionally use SCHEDJUX.

Finally, mixed coefficients test both aspects; however, the scores of all algorithms are higher due to an imprecise lower bound. GREEDYDEDICATED using BESTSCHEDULE dominates other algorithms with medians 1.46 for small instances and 1.22 for large ones. While the numerical values are higher, we still clearly see the advantage of using type-aware algorithms, as SCHEDMIXED (used without GREEDYDEDICATED) has a significantly higher median score (1.77 for small instances, 1.91 for large).

Due to space constraints, we do not present results in function of the number of tasks or the number of types. However, we have not found any strong dependencies between these variables and the results of our algorithm (apart from slightly — up to 1.18 — higher medians for 500 and 1000-task instances for compatible coefficients, caused by the LP problems discussed above).

Our results clearly show that using $P||C_{\max}$ algorithms without regarding types (SCHEDMIXED) is dominated by approaches considering types: using dedicated machines for $\alpha > 1$ or, in some $\alpha < 1$ instances, merging schedules of different types.

7 Related Work

We introduced the side-effects performance model [18], 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 allows 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 one data of the problem (number of types, and either the number of machines or the number of admissible sizes of the tasks).

Alternative models of data center resource management. A recent survey is [20]. Many colocation performance models are too complex for combinatorial results [11, 16, 17]. Schedulers rely on heuristic approaches with no formal performance guarantees [3,4,10,26]. In *bin-packing* approaches (e.g., [23,25]), tasks are modeled as items to be packed into bins (machines) of known capacity [5]. 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, memory, disk or network bandwidth) [24]. Alternatively, if tasks have unit-size requirements, simpler representations can be used, such as maximum weighted matching [1]. Bin packing 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.

Statistical approaches. Bobroff et al. [2] 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. [6] 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 [12, 14, 27]. [21] analyze the performance of colocated CPU-intensive benchmarks; and [13] measures performance of colocated HPC applications. Our $\alpha_{t,t'}$ coefficients are similar to their interference/affinity metrics. Additionally [13] shows a greedy allocation heuristics, but they don't study its worst-case performance nor the complexity of the problem.

8 Conclusion

We considered a problem of optimally allocating 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' is their total load times a coefficient $\alpha_{t',t}$, describing how compatible is t' with performance of t. 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 (we prove their approximation ratios for two types in the accompanying

technical report [19]). 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 results for 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.

We are also working on validating our model by a systems study. We are developing an extension for kubernetes that collects and correlates performance metrics reported by containers to derive the size/coefficient performance model.

Acknowledgements We thank Pawel Janus for his help in processing the Google cluster data. This research has been partly supported by a Google Faculty Research Award, a Polish National Science Center grant Sonata (UMO-2012/07/D/ST6/ 02440), 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. Beaumont, O., Eyraud-Dubois, L., Thraves Caro, C., Rejeb, H.: Heterogeneous resource allocation under degree constraints. IEEE TPDS 24(5) (2013)
2. Bobroff, N., Kochut, A., Beaty, K.: Dynamic placement of virtual machines for managing SLA violations. In: IM, Proc. IEEE (2007)
3. Bu, X., Rao, J., Xu, C.z.: Interference and locality-aware task scheduling for mapreduce applications in virtual clusters. In: HPDC, Proc. ACM (2013)
4. Chiang, R.C., Huang, H.H.: Tracon: interference-aware scheduling for data-intensive applications in virtualized environments. In: SC Proc. ACM (2011)
5. Coffman Jr, E.G., Garey, M.R., Johnson, D.S.: Approximation algorithms for bin packing: A survey. In: Approximation algorithms for NP-hard problems. PWS (1996)
6. Di, S., Kondo, D., Wang, C.: Optimization of composite cloud service processing with virtual machines. ToC (2015)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness (1979)
8. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAP 17(2) (1969)
9. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems theoretical and practical results. JACM 34(1), 144–162 (1987)
10. Jersak, L.C., Ferreto, T.: Performance-aware server consolidation with adjustable interference levels. In: SAC Proc. (2016)
11. 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)
12. Kambadur, M., Moseley, T., Hank, R., Kim, M.A.: Measuring interference between live datacenter applications. In: SC, Proc. IEEE (2012)
13. 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: CCGrid Proc. IEEE CS (2015)

14. Koh, Y., Knauerhase, R., Brett, P., Bowman, M., Wen, Z., Pu, C.: An analysis of performance interference effects in virtual environments. In: ISPASS, Proc. IEEE (2007)
15. Koutsoupias, E., Papadimitriou, C.: Worst-case equilibria. In: STACS Proc. Springer (1999)
16. Kundu, S., Rangaswami, R., Dutta, K., Zhao, M.: Application performance modeling in a virtualized environment. In: HPCA. IEEE (2010)
17. Kundu, S., Rangaswami, R., Gulati, A., Zhao, M., Dutta, K.: Modeling virtualized applications using machine learning techniques. In: SIGPLAN Not. vol. 47. ACM (2012)
18. Pascual, F., Rzadca, K.: Partition with side effects. In: HiPC 2015, Procs. (2015)
19. Pascual, F., Rzadca, K.: Optimizing egalitarian performance in the side-effects model of colocation for data center resource management. CoRR abs/1610.07339v2 (2017), `http://arxiv.org/abs/1610.07339`
20. Pietri, I., Sakellariou, R.: Mapping virtual machines onto physical machines in cloud computing: A survey. CSUR 49(3) (2016)
21. 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: CCGrid Proc. (2015)
22. Reiss, C., Tumanov, A., Ganger, G.R., Katz, R.H., Kozuch, M.A.: Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In: SoCC, Proc. ACM (2012)
23. Song, W., Xiao, Z., Chen, Q., Luo, H.: Adaptive resource provisioning for the cloud using online bin packing. IEEE ToC 63(11) (2014)
24. Stillwell, M., Vivien, F., Casanova, H.: Virtual machine resource allocation for service hosting on heterogeneous distributed platforms. In: IPDPS Procs. IEEE (2012)
25. Tang, X., Li, Y., Ren, R., Cai, W.: On first fit bin packing for online cloud server allocation. In: IPDPS Proc. (2016)
26. Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., Wilkes, J.: Large-scale cluster management at Google with Borg. In: EuroSys Proc. ACM (2015)
27. Xu, Y., Musgrave, Z., Noble, B., Bailey, M.: Bobtail: Avoiding long tails in the cloud. In: NSDI, Proc. (2013)