View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by INRIA a CCSD electronic archive server

Column generation integer programming for allocating jobs with periodic demand variations

Ikbel Belaid, Lionel Eyraud-Dubois

▶ To cite this version:

Ikbel Belaid, Lionel Eyraud-Dubois. Column generation integer programming for allocating jobs with periodic demand variations. International Workshop on Algorithmic Aspects of Cloud Computing, Sep 2015, Patras, Greece. hal-01252770

HAL Id: hal-01252770

https://hal.inria.fr/hal-01252770

Submitted on 8 Jan 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Column generation integer programming for allocating jobs with periodic demand variations

Ikbel Belaid and Lionel Eyraud-Dubois

[1] Inria Bordeaux – Sud-Ouest
[2] University of Bordeaux
Ikbel.Belaid@inria.fr, Lionel.Eyraud-Dubois@inria.fr

Abstract. In the context of service hosting in large-scale datacenters, we consider the problem faced by a provider for allocating services to machines. An analysis of a public Google trace corresponding to the use of a production cluster over a long period shows that long-running services experience demand variations with a periodic (daily) pattern, and that services with such a pattern account for most of the overall CPU demand. This leads to an allocation problem where the classical Bin-Packing issue is augmented with the possibility to co-locate jobs whose peaks occur at different times of the day, which is bound to be more efficient than the usual approach that consist in over-provisioning for the maximum demand. In this paper, we propose a column-generation approach to solving this problem, where the subproblem uses a sophisticated SOCP (Second Order Cone Program) formulation. This allows to explicitely select jobs which benefit from being co-allocated together. Experimental results comparing with theoretical lower bounds and with standard packing heuristics shows that this approach is able to provide very efficient assignments in reasonable time.

1 Introduction

The Cloud paradigm provides an illusion of infinite elasticity and seamless provisioning of IT resources. However, as providers keep scaling their infrastructures year after year, the efficient allocation of services in *Platform-as-a-Service* (PaaS) becomes crucial.

We concentrate on the case of a Cloud platform in which several independent services, typically virtualized as Virtual Machines (VMs) or lightweight containers, are serving user queries and need to be allocated onto physical machines (PMs) [19,1]. We consider the static case where a set of *dominant* services define the overall resource usage of the physical platform, which has proved to be commonplace in large datacenters[3]. In this context, mapping services with heterogeneous computing demands onto PMs is amenable to a multi-dimensional Bin-Packing problem (each dimension corresponding to a different kind of resource, memory, CPU, disk, bandwidth,. . .). Indeed, on the infrastructure side, each physical machine presents a given computing capacity (*i.e.* the number of Flops it can process during one time-unit), a memory capacity and a failure rate

(*i.e.* the probability that the machine will fail during the next time period). On the client side, each service has a set of requirements along the same dimensions (memory and CPU footprints) and a reliability requirement that has been negotiated typically through an SLA [9].

In this work, we consider a specific feature of CPU demand that arises in the context of service allocation. Previous work on the subject [4] argues that many services representing most of the overall CPU demand exhibit daily patterns and their demand can be modeled as a set of sinusoids, each comprising a constant component, an amplitude and a phase. This premise gives rise to a model for jobs with time-varying resource demands and to the associated packing problem. Such a model can be used to aggregate onto the same physical machines more resources than it would be possible based on their maximal demands only, taking advantage of the fact that different phases for different services imply that peak demands do not occur simultaneously. In this paper, we propose an algorithm based on column generation for packing jobs with periodic demands on the hosting platform. This algorithm provides very efficient allocations, compared to state-of-the-art greedy packing heuristics.

The remaining of this paper is organized as follows. We discuss some related works in Section 2. In Section 3, we present the formulation of the optimization problem as a *Second Order Cone Program* (SOCP). In Section 4, we propose our efficient packing algorithm based on column generation, whose performance is analyzed and validated on realistic and simulated data in Section 5. Finally, conclusions are drawn in Section 6.

2 Related works

In order to deal with resource allocation problems arising in the context of Clouds, several sophisticated techniques have been developed in order to optimally allocate user services onto PMs, either to achieve good load-balancing [8,5] or to minimize energy consumption [6]. Most of the approaches in this domain are based on offline [10] and online [11] variants of Bin-Packing strategies.

In this paper, we concentrate on the allocation of jobs that last for a long time and whose CPU demands exhibit periodic patterns. Some other work deal with allocating jobs whose demands varies over time, either with predictable (static) or unknown (dynamic) behavior. In the static case which is the focus of this present work, historical average resource utilization is typically used as input to an algorithm that maps services to physical machines. Therefore, the mapping is done off-line. In contrast, dynamic allocation schemes are implemented on shorter timescales. Dynamic allocation leverages the ability to perform runtime migrations of jobs and to recompute resource allocation amongst services. A dynamic migration algorithm *Measure Forecast Remap* is introduced in [7], where highly variable workloads are forecast over intervals shorter than the time scale of demand variability to ensure dynamic execution minimization of the number of required machines. Based on stochastic vector packing model, the static scheme proposed in [15] makes use of customers' periodic access patterns

in web server farms to assign each customer to a server so as to minimize the total number of required servers. In this latter work, the variable demand is analyzed at a different time scale to extract probability distributions that are independent of time. Then, *stream-packing* heuristics are employed to select the most complementary jobs to be packed in the same server. Urgaonkar et al. [16] rely on on-line application profiling to demonstrate the feasibility and benefits of overbooking resources in shared platforms to guide the application placement onto dedicated resources while providing performance guarantees at runtime. A new mechanism for dynamic resource management in cluster-based network servers [2], called cluster reserve, allows performance isolation between service classes and provides a minimal amount of resources, irrespective of the load imposed by other requests. In contrast to these other directions, our work focuses on a part of the workload which exhibits deterministic periodic variability. In this context, dynamic resource management is unnecessary: the migration cost can be avoided by using periodicity-aware static approaches for service allocation. By focusing on long-running services with high workloads, it is possible to apply sophisticated techniques to provide efficient packing policies, which results in increased resource usage. Smaller or short-lived jobs, which are much more numerous but represent a smaller part of the resource usage, can be handled with usual greedy allocation schemes.

This paper is a followup to [4], which analyzes the performance of several standard packing heuristics in the context of packing jobs with periodic demand variation. In this paper, we propose the use of the *Dantzig-Wolfe* decomposition [17] to solve very efficiently the corresponding packing problem. In fact, mathematical programs featured by a large space of integer variables are particularly suited for *Dantzig-Wolfe* decomposition that reformulates the original compact problem to provide a tighter linear programming relaxation bound. This decomposition relies on delayed column generation algorithm. The overarching idea of this algorithm is that many programs are too large to consider all the variables explicitly. Since most of the variables will be neglected in the optimal solution, only a subset of variables need to be considered in theory when solving the problem. Column generation leverages this idea to generate only the variables which have the potential to improve the objective function, that is, to find variables with negative reduced costs. Section 4 details the utilisation of *Dantzig-Wolfe* decomposition to reformulate the packing of jobs with variable demands on hosted parallel machines based on the original formulation and employing the column generation algorithm.

The *Dantzig-Wolfe* reformulation gives rise to a master problem and subproblems, whose typically large number of variables is dealt with implicitly by using an integer programming column generation procedure, also known as branch-and-price algorithm. Solving the master problem does not require an explicit enumeration of all its columns because the column generation algorithm allows one to generate columns if and when needed. In many cases, this allows huge integer programs that had been previously considered intractable to be solved. The technique of *Dantzig-Wolfe* using the approach of column genera-

tion has been applied successfully in many classical problems as: cutting stock, vehicle routing, crew scheduling, etc.

3 Packing of jobs with periodic demands

An analysis of a publicly available Google trace [14,13] has shown that about two-thirds of the dominant, normal production jobs in that trace exhibit significant daily pattern [3]. Based on this analysis, and following [4], we consider a packing problem for those long running jobs, which account for a large portion of the workload.

3.1 Notations and problem formulation

Let us assume that the cloud platform we consider consists of M homogeneous nodes $M_1, \ldots, M_k, \ldots, M_M$ and let us denote the processing capacity of a node by C. For the sake of simplicity and in order to focus on issues related to the aggregation of periodic demands, we will concentrate on CPU demands only. The tasks of a job (corresponding to a service in the trace) can run on any node, and job J_j is split into N_j tasks denoted by $T_{j,1}, \ldots, T_{j,l}, \ldots, T_{j,N_j}$, who share the same characteristics in terms of CPU demand.

In turn, platform nodes are allowed to run several tasks, provided that at any time, their capacity is not exceeded. We assume that the set of tasks running on a node does not change over time, what is a realistic assumption for dominant *Normal Production* jobs, and we model the instantaneous demand at time t of task $T_{j,l}$, which does not depend on l, as

$$W_j(t) = C_j + \rho_j \sin\left(2\pi \frac{t}{P_j} + \phi_j\right)$$

where C_j denotes the average of CPU demand of Task $T_{j,l}$, ρ_j denotes its maximal amplitude with respect to C_j, P_j denotes its period and which is common for all the jobs. In the remainder of the paper, job period will be named P. ϕ_j denotes its phase.

In this context, our aim is to provide a static packing for the set of tasks $T_{j,l}$ such that at any step and on any resource, capacity constraints are not exceeded and such that the number of required nodes is minimized. More specifically, this model allows to take advantage of daily variations in order to obtain an efficient packing of tasks. Indeed, most packing strategies are based on the maximal demand of each task, what corresponds to $C_j + \rho_j$ for a task of job j. Taking advantage of the fact that all tasks do not achieve their peak demand at the same time in the day, it is possible to pack more tasks, and therefore to use fewer nodes whilst packing statically all the tasks.

The corresponding capacity constraint for a given machine M_k is thus

$$\forall t, \sum_{j,l:\ T_{j,l} \in M_k} W_j(t) \leq C,$$

and it can be rewritten [4] as an expression which does not depend on t:

$$\forall k, \quad \sum_{j,l:\ T_{j,l} \in M_k} C_j + \sqrt{(\sum_{j,l:\ T_{j,l} \in M_k} \rho_j \cos(\phi_j))^2 + (\sum_{j,l:\ T_{j,l} \in M_k} \rho_j \sin(\phi_j))^2} \leq C \tag{1}$$

This modified packing constraint yields a quadratically constrained programming (QCP) formulation of the problem. This formulation uses two types of variables: integer variables $X_{j,k}$ representing the number of tasks of job j allocated on the node M_k, and boolean variables Y_k representing whether node N_k is used. With these variables, the formulation is the following:

$$\text{Minimize} \sum_k Y_k$$

$$\forall j \in J, \quad \sum_{k \in M} X_{j,k} = N_j \tag{2}$$

$$\forall k \in M, \quad (\sum_{j \in J} X_{j,k}\ \rho_j\ \cos(\phi_j))^2 + (\sum_{j \in J} X_{j,k}\ \rho_j\ \sin(\phi_j))^2 \leq (C\ Y_k - \sum_{j \in J} X_{j,k}\ C_j)^2 \tag{3}$$

$$\forall k \in M, \quad C\ Y_k - \sum_{j \in J} X_{j,k}\ C_j \geq 0 \tag{4}$$

In this formulation, constraint (2) ensures that all instances of all jobs are allocated. Tasks belonging to the same job could co-exist in the same node. Constraints (3) and (4) are a quadratic reformulation of Equation (1), ensuring that an unused node does not contribute any resource to the platform. Due to the nature of this constraint, this formulation can be expressed as a Second Order Cone Program (SOCP) [12], and can thus benefit from efficient general purpose solvers [12] for convex optimization. However, as noticed in [4], on real-size instances with thousands of machines, this formulation can not be solved in reasonable time with integer and boolean values. Relaxing the problem by allowing rational variables makes it possible to obtain a lower bound on the necessary number of resources in reasonable time.

In the next Section, we describe how to reformulate this problem with a *Dantzig-Wolfe* decomposition, which allows to quickly obtain very good solutions to the packing problem.

4 Integer programming Column generation

Dantzig-Wolfe decomposition has been an important tool to solve large structured models that could not be solved using standard algorithms as they exceeded the capacity of solvers. The main idea behind this technique is to decompose the original problem into a number of independent subproblems, whose solutions are then assembled by solving a so-called master problem. This master

problem is then solved iteratively. In our case, we can identify the natural decomposition of the problem: for two different values of k (i.e. for each machine), the corresponding sets of constraints (3) and (4) are independent, because they contain disjoint sets of variables. Since we assume that machines are homogeneous, all those subproblems are actually identical, and we obtain a special case where solving it only once is enough.

In the *Dantzig-Wolfe* reformulation, we obtain a master problem which contains one variable for each solution to this subproblem. In our case, such a solution is simply a valid packing configuration for a machine, i.e. a set of jobs which can be allocated together on a single machine while respecting the capacity constraint. One configuration Z_i can be represented as a J-uplet $(X_{1,i}, X_{2,i}, \ldots, X_{J,i})$, where $X_{j,i}$ is the number of tasks of job j in configuration i. As discussed previously, this configuration is valid if it satisfies equation (1).

In the following, we will denote as K the set of all valid configurations. The master problem contains one variable Y_i for each configuration $Z_i \in K$, which represents the number of machines which use the configuration Z_i, i.e. the number of machines to which $X_{j,i}$ tasks of job j are allocated. The packing problem can now be formulated as follows:

Master Problem: Minimize $\sum_{i \in K} Y_i$ s.t

$$\forall j \in J, \quad \sum_{i \in K} X_{j,i} \; Y_i \geq N_j \tag{5}$$

$$Y_i \in \mathbb{N} \tag{6}$$

This master problem cannot be solved directly due to an exponential number of variables. However, the column generation approach consists in considering variables only from a subset $K' \subset K$, and to solve the resulting *restricted master problem* (RMP) on this set of variables.

This restricted problem may provide a sub-optimal solution, since there might exist a configuration in $K \backslash K'$ which improves the solution. In order to find this configuration, one can write the dual of the master problem, in which there is one variable π_j for each job, and one constraint for each configuration:

Dual Master Problem: Maximize $\sum_{j \in J} N_j \pi_j$ s.t

$$\forall i \in K, \quad \sum_{j \in J} X_{j,i} \; \pi_j \leq 1 \tag{7}$$

$$\pi_j \geq 0 \tag{8}$$

The sub-optimal solution obtained from the restricted master problem provides a solution π_j^* of the dual master problem which is possibly infeasible, since not all constraints are included in the dual of the RMP. From this solution π_j^*, we can identify a variable to add to the problem by searching for a violated constraint in the dual, i.e. a configuration $Z_i \in K$ such that $\sum_{j \in J} X_{j,i} \; \pi_j^* > 1$. This gives rise to the following subproblem, with one variable U_j for each job:

Subproblem: Periodic Knapsack: Minimize $1 - \sum_{j \in J} \pi_j^* U_j$ s.t

$$\forall j \in J, \quad (\sum_{j \in J} U_j \ \rho_j \ \cos(\phi_j))^2 + (\sum_{j \in J} U_j \ \rho_j \ \sin(\phi_j))^2 \leq (C - \sum_{j \in J} U_j \ C_j)^2 \quad (9)$$

$$U_j \in \mathbb{N} \quad (10)$$

If the optimal solution of this subproblem has a negative value, then we have identified a configuration to add to the RMP. On the other hand, if this problem has no solution with negative value, it means that all constraints in the dual of the master problem are satisfied with the current solution, which implies that this current solution of the RMP is actually optimal for the master problem.

This subproblem can be seen as a knapsack problem: given profits π_j^* for each job, we search for the set of tasks with maximal profit which can fit in a single machine. The strong point of the *Dantzig-Wolfe* reformulation in this case is that we have isolated the quadratic constraint in the subproblem, which is of much smaller scale than the original problem, with only one variable per job. This problem can now be solved (quite efficiently as we will see in Section 5) with a general integer SOCP solver.

The column generation algorithm is summarized in Algorithm 1: starting from an initial set of configurations (which we describe below), the algorithm iteratively solves the RMP, and then uses the values of the dual variables as prices in the knapsack subproblem. The solution to this subproblem yields a new configuration which is added to the set, and a new iteration is performed. The process ends when no solution to the subproblem has a negative cost. The obtained RMP is then solved with integer constraints to obtain a feasible solution to the original problem. In practice, this last step is often too time consuming for the solver to obtain an optimal solution in reasonable time, so in our experimental evaluation, we included a 5 minute time limit and use the best feasible solution obtained by the solver in that time.

The initial set of configurations can be chosen arbitrarily, as long as the first RMP is feasible, i.e. all jobs are represented in at least one configuration. For simplicity, we build the initial set K_0 with one configuration per job, where the configuration for job j contains $\left\lfloor \frac{C}{C_j + \rho_j} \right\rfloor$ tasks of job j (as many as can fit on one machine), and 0 tasks of all other jobs.

Data: Job characteristics: C_j, ρ_j, ϕ_j and N_j
Result: Feasible solution: a set K_t of configurations and values $(Y_i)_{i \in K_t}$ stating how many machines use each configuration

```
t ← 0
K_t ← K_0
repeat
    Solve RMP with variables in K_t
    Generate dual values π*_j from the RMP solution
    Solve the subproblem with prices π*_j
    if strictly negative reduced cost then
        Col ← new column with the coefficients of the subproblem solution
        t ← t + 1
        K_t ← K_{t-1} ∪ Col
    end
until no negative reduced cost solution;
Solve RMP with variables in K_t as an integer program
```

Algorithm 1: Column generation algorithm

5 Experimental evaluation

In this section, we present the results of synthetic and realistic experiments provided by column-generation algorithm and best-effort heuristics. We investigate the performance of each in reducing the number of used nodes as well as their margin towards the lower bound. We show that our column generation algorithm delivers good results in reducing the number of iterations and computation time.

5.1 Complexity and Lower Bound

The optimization problem that consists in packing tasks with periodic demands into nodes is clearly NP-Complete, since it is amenable to classical Bin-Packing problems [10,11] in its most simplified setting where $\forall j,\ \rho_j = 0$, *i.e.* the case when demands do not change over time. Indeed, in many cases the last step of our column generation algorithm (where we look for an integer solution) is unable to obtain an optimal solution in the alloted time. In order to asses the performance of the obtained results, we rely on a simple but powerful lower bound: the total workload at time t is $\sum_{j \in J} W_j(t)$, whose peak can be computed like in Section 3 as $W = \sum_{j \in J} C_j + \sqrt{(\sum_{j \in J} \rho_j \cos(\phi_j))^2 + (\sum_{j \in J} \rho_j \sin(\phi_j))^2}$. Since in any solution, the sum of the capacity used on each machine is not lower than P, we know that any solution must use at least $\lceil \frac{W}{C} \rceil$ machines. This solution is not feasible in general but it provides a lower bound on the number of necessary nodes.

5.2 Heuristics

In this Section, we present some heuristics introduced in [4], adapted from classical efficient greedy Bin-Packing algorithms to the case of tasks exhibiting daily

patterns. In the following, we denote by $\mathcal{L}(M_k, T_{j,l})$ the peak load on machine M_k after adding one task $T_{j,l}$ of job J_j to M_k.

- Best-Fit Decreasing $\mathcal{BFD}$ is a greedy algorithm in which tasks are considered by decreasing values of C_j. At any step, task $T_{j,l}$ (from job J_j) is allocated to the node M_k such that $\mathcal{L}(M_k, T_{j,l})$ is maximized (while remaining below C). Note that contrarily to what happens in classical $\mathcal{BFD}$, the size that is considered is the size after the allocation. If no such node exists, then a new node is added to the system to hold the task.
- In Min-Max $\mathcal{MM}(M)$, the target number of nodes is fixed to M a priori. Then, $\mathcal{MM}$ is a greedy algorithm where tasks are considered by decreasing values of C_j. At any step, task $T_{j,l}$ (from job J_j) is allocated to the node M_k such that $\mathcal{L}(M_k, T_{j,l})$ is minimized, in order to balance the load between the different nodes. The allocation may fail if M is too small. We thus use dichotomic search to find the smallest value of M which allows to obtain a solution.
- Min-Max-Module $\mathcal{MMM}$ is similar to $\mathcal{MM}$, except that tasks are represented using their maximal demand over time $C_j + \rho_j$ only. This is typically what happens when one neglects the possibility to take advantage of the fact that peak demands do not occur at the same time for all jobs.

5.3 Simulated synthetic Data

First, we perform a set of experiments with synthetic data in order to assess the influence of the parameters on the performance of the different proposed methods. In all the experiments, we set the capacity of the nodes to 20, and we display the ratio between the number of nodes provided by the corresponding method against the lower bound on the number of necessary nodes described in Section 5.1.

For synthetic jobs, we consider the following parameters:

- CPU footprint of the tasks: we consider the case of Large Tasks, Medium Tasks and Small Tasks where C_j is chosen uniformly at random respectively in $[0, 10]$, $[0, 5]$ and $[0, 1]$. To keep the total workload constant, we set the number of tasks N_j in each job to 50 for Large Tasks, 100 for Medium Tasks, and 500 for Small Tasks.
- Daytime amplitude: we consider the case of Large Daytime Amplitude (where ρ_j is chosen uniformly at random in $[0, C_j]$) and Small Daytime Amplitude (where ρ_j is chosen uniformly at random in $[0, \frac{C_j}{2}]$).

In all cases, the phase of each job is chosen uniformly at random in $[0, 2\pi[$, and the number of jobs is set to 100. We performed other experiments with different number of jobs and tasks, but the results showed very little sensitivity to these parameters and were excluded from the paper in order to save space. For each scenario, we have performed 20 experiments, and the results are shown on Figure 1, where the 20 experiments for each scenario and each algorithm are

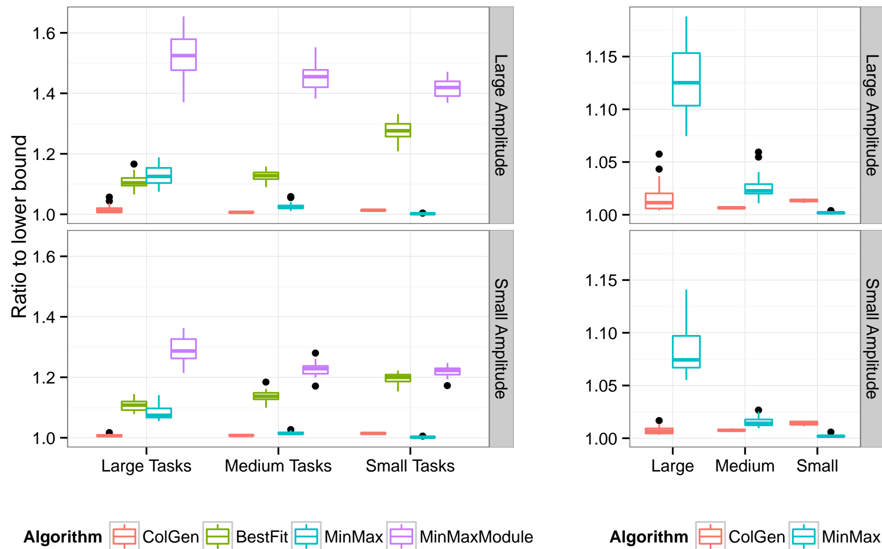

Fig. 1: Performance of all algorithms on synthetic data. The right plot is a focus on the most efficient algorithms.

grouped together in a boxplot showing the mean, the first and third quantiles, and minimum and maximum values.

The figure shows that the column generation algorithm is able to consistently provide solutions with a number of required nodes very close to the lower bound, in all of the scenarios. Actually, for some of the Large Tasks scenarios , the solver is able to obtain a provably optimal integer solution in the final step of the algorithm, meaning that the column generation algorithm actually provides an optimal solution in these cases. This also shows that the lower bound is actually very precise, since it is possible to exhibit a feasible solution with very close performance.

In the Small Tasks scenarios, in which each node can hold a few tens of tasks, Min-Max $\mathcal{MM}$ performs extremely well and is always at most within 1% of the lower bound. This behaviour is usual in Bin-Packing problems: the presence of very small objects makes the packing easier since they can be used to fill the wasted space in the bins. Indeed, the results of Min-Max $\mathcal{MM}$ degrade when tasks get larger: in this case, the number of tasks per node is relatively small (a few units) and greedy heuristics fail to achieve close to optimal performance. On the other hand, in the Small Tasks scenarios, the solution provided by the column generation algorithm is not as good. This comes from the fact that the integer problem of the final step is very difficult to solve in that case.

Nevertheless, the number of nodes required by the column generation algorithm is always within 1% of the lower bound. It is worth noting that in the

context which we consider in this work (long-running services with heavy workload), the task sizes are not small, and the medium-large task sizes are the most realistic cases (see Section 5.4 for a comparison on a real trace).

The BestFit heuristc $\mathcal{BFD}$ represents the standard packing algorithm used in such Cloud systems. We can see that its performance is consistently 10% above the lower bound , and even worse in the case of small tasks. This advocates strongly in favor of more sophisticated algorithms like the one we propose. Finally, we can also see that failing to take periodic demand variations leads to a large waste of resources. Indeed, the performance of Min-Max-Module $\mathcal{MMM}$ is consistently far from the lower bound, by 50% in the case of Big Amplitudes and by 25% in the case of Small Amplitudes.

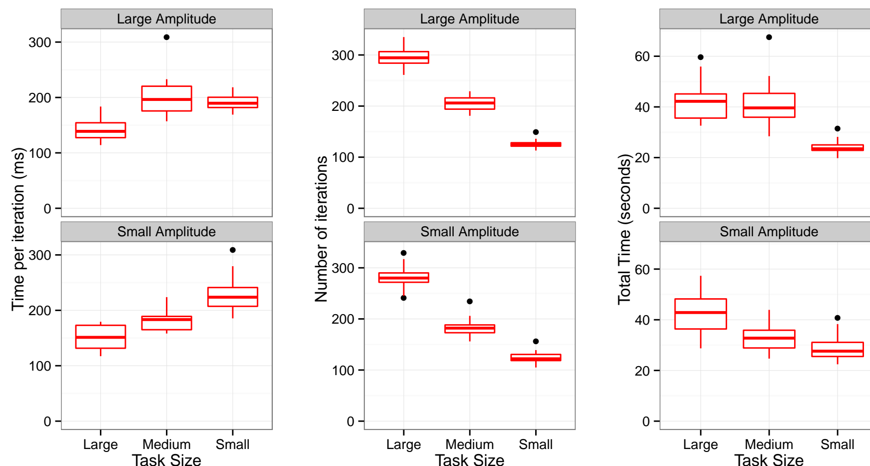

Fig. 2: Running time of column generation algorithm.

Figure 2 analyses the computation time for the first phase of the column-generation algorithm, which solves the rational relaxation of the problem (the second phase is the final step where we obtain an integer feasible solution, whose time is limited to 5 minutes in our experiments). We observe that the number of iterations remains below 300, and the time per iteration is very low (around 200ms), showing that the SOCP formulation for the subproblem is very efficient. This allows the column generation algorithm to complete its first phase in about 40 seconds in all scenarios.

5.4 Jobs and Tasks of Google Trace

Then, we concentrate on the set of realistic periodically variable jobs in the trace released by Google [18] and corresponding to one production center. In [4],

	Column generation $\mathcal{CG}$	Best-Fit $\mathcal{BFD}$	Min-Max $\mathcal{MM}$	Min-Max-Module $\mathcal{MMM}$
Number of Nodes	2103	2182	2114	2226

Table 1: Number of nodes required per heuristic.

a instance has been extracted from this trace with 89 jobs corresponding to a total of 22600 tasks. The largest job in terms of tasks consists of 1608 tasks and the largest job in terms of CPU demand corresponds to the capacity of 184 nodes at its peak demand. A capacity equivalent to 2198 nodes would be required if all jobs reached their peak demand at the same instant. The overall peak demand for the whole set of jobs is equivalent to the capacity of 2090 nodes. Therefore, there exists a potential improvement on the number of required nodes of 5%, what should be considered as large in the context of an actual production center. We have applied our column-generation algorithm on this instance, and the results achieved are displayed in Table 1.

The results of $\mathcal{MM}$ are deemed extremely good in [4], because the number of required machines is only 1.1% higher than the lower bound. Our column-generation algorithm $\mathcal{CG}$ is able to provide an even more efficient solution, with a number of nodes only 0.6% higher than the lower bound, effectively halving the gap between the best solution and the lower bound. As shown previously, the time complexity of our algorithm is very reasonable, showing that our column generation algorithm can really make an impact for improving resource usage in actual production centers.

6 Conclusions

Allocating computing resources for multiple time-varying job workloads is an attractive yet challenging target for many providers of large-scale infrastructures of cloud computing. Towards this end, we address in this paper a resource allocation problem for jobs that exhibit daily periodic sinusoidal patterns. Such jobs have been shown to represent a significant part of the workload of large production clusters, as exemplified by a trace from a Google center. Taking the periodic pattern into account allows to coallocate jobs which reach their peaks at different times, and this allows to to significantly increase the resource usage on these platforms.

In this paper, we present a novel packing technique relying on job aggregation mechanism by employing an exact method using column generation integer programming. The *Dantzig-Wolfe* reformulation allows to isolate the quadratic constraint in a small size subproblem, which can be solved very efficiently. Solving iteratively this subproblem and the reformulated packing problem allows to efficiently identify the relevant machine configurations, i.e. the set of jobs which should be allocated together. This technique is then compared to best-effort heuristics inspired from the standard bin-packing methods, on both simulated and realistic data. Experimental results show that this algorithm obtains good

results very consistently, even in difficult cases in which task sizes are large and few tasks can fit together on the same machine. In the most realistic cases, the column generation improves over the best heuristic by up to 5%, effectively halving the gap between the best known solutions and the lower bound.

As future work, we plan to extend job aggregation strategies to provide performance guarantees for other resources like memory, disk, network bandwidth, etc. Improving the column generation algorithm could focus on two different directions: using a more efficient routine to solve the subproblem could lower further the running time of the first phase, and more efficient branching schemes could improve the efficiency of the last step of the algorithm. Besides, we target to address the problem of resource allocation and sharing for dynamically arriving jobs while considering the already assigned static ones. This problem is challenging and attractive computing paradigm in cloud computing for a wide variety of applications. This dynamic co-allocation for unpredictable jobs presents new challenges to resource management in multicluster systems, such as locating sufficient resources for these dynamic jobs in distributed sites, managing temporarily the job assignment and coordinating their executions with the processing of the static jobs.

References

1. M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, et al. Above the clouds: A berkeley view of cloud computing. *University of California, Berkeley*, 2009.
2. Mohit Aron, Peter Druschel, and Willy Zwaenepoel. Cluster reserves: A mechanism for resource management in cluster-based network servers. In *In Proceedings of the ACM SIGMETRICS Conference*, pages 90–101, 2000.
3. O. Beaumont, L. Eyraud-Dubois, and J.-A. Lorenzo-del Castillo. Analyzing real cluster data for formulating allocation algorithms in cloud platforms. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2014 IEEE 26th International Symposium on*, pages 302–309, Oct 2014.
4. Olivier Beaumont, Ikbel Belaid, Lionel Eyraud-Dubois, and Juan-Angel Lorenzo-Del-Castillo. Allocating jobs with periodic demand variations. Accepted to EuroPar 2015, February 2015.
5. Olivier Beaumont, Lionel Eyraud-Dubois, Hejer Rejeb, and Christopher Thraves. Heterogeneous Resource Allocation under Degree Constraints. *IEEE Transactions on Parallel and Distributed Systems*, 2012.
6. A. Beloglazov and R. Buyya. Energy efficient allocation of virtual machines in cloud data centers. In *IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 577–578. IEEE, 2010.
7. N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing sla violations. In *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*, pages 119–128, May 2007.
8. R.N. Calheiros, R. Buyya, and C.A.F. De Rose. A heuristic for mapping virtual machines and links in emulation testbeds. In *Proceedings of International Conference on Parallel Processing (ICPP)*, pages 518–525. IEEE, 2009.
9. Walfredo Cirne and Eitan Frachtenberg. Web-scale job scheduling. In *Job Scheduling Strategies for Parallel Processing*, pages 1–15. Springer, 2013.

10. M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.
11. D. Hochbaum. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1997.
12. Hans D Mittelmann. An independent benchmarking of sdp and socp solvers. *Mathematical Programming*, 95(2):407–430, 2003.
13. Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Towards understanding heterogeneous clouds at scale: Google trace analysis. Technical Report ISTC–CC–TR–12–101, Intel science and technology center for cloud computing, Carnegie Mellon University, Pittsburgh, PA, USA, April 2012. Posted at `http://www.istc-cc.cmu.edu/publications/papers/2012/ISTC-CC-TR-12-101.pdf`.
14. Charles Reiss, John Wilkes, and Joseph L. Hellerstein. Google cluster-usage traces: format + schema. Technical report, Google Inc., Mountain View, CA, USA, November 2011. Revised 2012.03.20. Posted at URL `http://code.google.com/p/googleclusterdata/wiki/TraceVersion2`.
15. Johara Shahabuddin, Abhay Chrungoo, Vishu Gupta, Sandeep Juneja, Sanjiv Kapoor, and Arun Kumar. Stream-packing: Resource allocation in web server farms with a qos guarantee. In *High Performance Computing, HiPC 2001*, pages 182–191. 2001.
16. Bhuvan Urgaonkar, Prashant Shenoy, and Timothy Roscoe. Resource overbooking and application profiling in shared hosting platforms. *SIGOPS Oper. Syst. Rev.*, 36(SI):239–254, December 2002.
17. F. Vanderbeck. On dantzig-wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. *Operations Research*, pages 111–128, 2000.
18. John Wilkes. More Google cluster data. Google research blog, November 2011. Posted at `http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html`.
19. Q. Zhang, L. Cheng, and R. Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, 2010.