Allocating jobs with periodic demand variations

Olivier Beaumont, Ikbel Belaid, Lionel Eyraud-Dubois, Juan-Angel
Lorenzo-Del-Castillo

▶ To cite this version:

HAL Id: hal-01118176
https://hal.inria.fr/hal-01118176

Allocating jobs with periodic demand variations

Olivier Beaumont, Ikbel Belaid, Lionel Eyraud-Dubois, and
Juan-Angel Lorenzo-del-Castillo

[1] Inria Bordeaux – Sud-Ouest
[2] University of Bordeaux

Abstract. In the context of service hosting in large-scale datacenters, we consider the problem faced by a provider for allocating services to machines. Based on an analysis of a public Google trace corresponding to the use of a production cluster over a long period, we propose a model where long-running services experience demand variations with a periodic (daily) pattern and we prove that services following this model acknowledge 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 provide a mathematical framework to analyze the packing of services exhibiting daily patterns and whose peaks occur at different times. We propose a sophisticated SOCP (Second Order Cone Program) formulation for this problem and we analyze how this modified packing constraint changes the behavior of standard packing heuristics (such as Best-Fit or First-Fit Decreasing). We show that taking periodicity of demand into account allows for a substantial improvement on machine utilization in the context of large-scale, state-of-the-art production datacenters.

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) [17,1]. 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 [8].

In this work, we consider a specific feature of CPU demand that arises in the context of service allocation. Based on the analysis of a large cluster trace provided by Google, we demonstrate in Section 3 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. Under this premise, the contribution of this paper is threefold. First, we propose and advocate a novel model for jobs with time-varying resource demands and we define the associated packing problem. This 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. Second, we show the benefits of antagonistic job aggregation, and how this can be used to improve the system performance. Third, we propose several algorithm for packing jobs with periodic demands on the hosting platform. The first one is based on a Second Order Cone Program (SOCP) formulation [11] whereas the others are adaptations of classical greedy packing heuristics.

The remaining of this paper is organized as follows. We discuss some related works in Section 2. In Section 3, we characterize the periodic behavior of some of the jobs in a cluster usage trace provided by Google. In Section 4, we formulate the optimization problem using Complex Analysis and we prove that it can be expressed as a SOCP (*Second Order Cone Program*). In Section 5, we propose several packing heuristics, whose performance is analyzed and validated on a realistic trace in Section 6. Finally, conclusions are drawn in Section 7.

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 [7,4] or to minimize energy consumption [5]. Most of the approaches in this domain are based on offline [9] and online [10] 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 [6], where highly variable workloads are forecast over intervals shorter than the time scale

of demand variability to ensure dynamic minimization of the number of required machines. Based on stochastic vector packing model, the static scheme proposed in [14] 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. [15] 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. Still, above mentioned approaches can be used in order to allocate at runtime all the tasks that do not exhibit daily sinusoidal patterns in their demand. Nevertheless, we will prove that the overall weight of such services in terms of CPU demand makes it useful to design specific allocation algorithms for them.

3   Periodicity analysis

When considering efficient allocations, it is important to categorize how services are correlated in order to schedule them efficiently. Indeed, if many services reach their (say, CPU) peak demand at the same time (*i.e.* high positive correlation), the stress on the platform and on the resource allocation algorithm will be much higher. In this case, it seems reasonable to place those services on different physical machines to avoid machine starvation. On the other hand, if peaks are spread on a large enough time-frame, this will allow for some slack in the allocation algorithm to provide efficient placements by co-allocating jobs whose peaks happen at different times, hence resulting in a more efficient average resource utilization.

Our periodicity analysis is based on the study of a usage trace released by Google from one of its production clusters [16]. The workload consists in a massive number of jobs, which can be further divided into *tasks*, being each task assigned to a single physical machine. The data are collected from 12583 machines, span a time of 29 days and provide exhaustive profiling information on 5-minute monitoring intervals. Each job belongs to a priority group, namely (in order of decreasing importance) *Infrastructure*, *Monitoring*, *Normal Production*, *Other* and *Gratis (free)*[13,12]. The scheduler generally gives preference to resource demands from higher priority tasks over tasks belonging to lower priority groups, to the point of evicting the latter ones if needed.

Given the thorough information contained in the trace, one of the main difficulties is related to the time needed to validate any assumption based on

these data. To simplify this process without loss of accuracy, we proposed in [3] an extraction of the information from a subset of jobs that we defined as *dominant*, *i.e.* jobs which account for most of platform usage at any time.

In this work, we have restricted our study to dominant jobs in the *Normal Production* class, given that they represent standard production utilization in the datacenter and last for long enough to allow periodicity correlation. In addition, considering only one priority class avoids issues due to the fact that hosts have finite capacity. Indeed, this finite capacity implies that when the resource demand of one job increases, another job with lower priority may end up using fewer resources (or even getting evicted by the scheduler) even if its actual demand remained invariable.

The spectral analysis of the *Normal Production*, dominant jobs that run during the whole trace allowed us to quantify the main components of their CPU demand, namely the amplitude, phase, frequency and background noise. Table 1 provides the averaged ratios between the jobs' components' amplitudes and their constant part. The residual noise is about 6% of the average CPU demand for a large part of the jobs, which can be used as a threshold: any pattern with an amplitude significantly larger can be identified as a relevant component. We conclude that very few jobs exhibit hourly patterns, more than half of the jobs exhibit very strong daily patterns, and only two thirds have significant daily patterns. Weekly patterns are not as strong, but they are still significant for about half of the jobs.

Regarding pattern synchronization, we observed that all jobs with a weekly pattern show the same behavior: 5 days of high usage followed by 2 days of lower usage. For the daily patterns, we analyzed jobs with an amplitude of, at least, 10% of the mean. In half of the jobs, the phase difference observed is below 60 degrees (*i.e.* their peaks are within 4 hours from each other). Furthermore, 90% of the jobs exhibit a phase difference below 120 degrees (*i.e.* peaks are at most 8 hours apart). This shows that the jobs' behavior is clearly correlated by this daily pattern.

Stats	Ratio of Amplitude to mean				
	Hourly	*Daily*	*Weekly*	*Long term*	*Noise*
mean	0.057	0.267	0.148	0.154	0.100
std	0.246	0.232	0.127	0.161	0.154
min	0.001	0.006	0.011	0.001	0.012
25%	0.004	0.052	0.076	0.051	0.036
50%	0.007	0.268	0.106	0.102	0.058
75%	0.009	0.376	0.196	0.196	0.072
max	1.612	1.075	0.669	1.149	0.836

Table 1: Ratios Amplitude/DC for long-running, dominant jobs[3].

4 Packing of jobs with periodic demands

4.1 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 step, 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, as shown in Section 3, 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 the period of its pattern and ϕ_j denotes its phase. As noticed in Section 3, one can concentrate in this context on jobs that exhibit daily patterns and we will therefore assume in what follows that $\forall j, P_j = P$, where P denotes a daytime.

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, our goal is 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.

Let us consider several tasks $T_{j,l}$ clustered together on node M_k. Knowing that all the jobs have the same period P, the constraint stating that the capacity of M_k is not exceeded at any time

$$\forall t, k, \quad \sum_{j,l,\ T_{j,l} \in M_k} W_j(t) \leq C, \ \text{becomes}$$

$$\iff \forall t, k, \quad \sum_{j,l,\ T_{j,l} \in M_k} C_j + \sum_{j,l,\ T_{j,l} \in M_k} \rho_j \sin(2\pi t/P + \phi_j) \leq C$$

$$\iff \forall t, k, \quad \sum_{j,l,\ T_{j,l} \in M_k} C_j + \mathrm{Im}\left(\sum_{j,l,\ T_{j,l} \in M_k} \rho_j \exp(2i\pi t/P)\ \exp(i\phi_j)\right) \leq C$$

$$\iff \forall t, k, \quad \sum_{j,l,\ T_{j,l} \in M_k} C_j + \mathrm{Im}\left((\exp(2i\pi t/P)) \left(\sum_{j,l,\ T_{j,l} \in M_k} \rho_j \exp(i\phi_j) \right) \right) \leq C$$

$$\iff \forall k, \quad \sum_{j,l,\ T_{j,l} \in M_k} C_j + \| \sum_{j,l,\ T_{j,l} \in M_k} \rho_j \exp(i\phi_j) \| \leq C,$$

where $Im(z)$ denotes the imaginary part of complex number z, i is the imaginary unit satisfying $i^2 = -1$ and $\|z\|$ denotes the modulus of z.

Note that in the last expression, the constraint does not involve t anymore, and that all above complex analysis derivations are equivalences, such that this last expression exactly states that the capacity constraint is never exceeded at any time step. In order to design exact solutions and heuristics, we will use the following formulation,

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

$$(1)$$

4.2 Quadratic formulation

From this modified packing constraint (1), we propose a quadratically constrained programming (QCP) formulation of our 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,

Boolean variables Y_k representing whether node N_k is used.

With these variables, the formulation is the following:

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

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

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

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

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, and can thus benefit from efficient general purpose solvers [11] for convex optimization. However, 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.

5 Packing Heuristics

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 [9,10] in its most simplified setting where $\forall j$, $\rho_j = 0$, *i.e.* the case when demands do not change over time. The SOCP formulation proposed in Section 4.2 can be used to solve the optimization problem, but its use is in practice restricted to small cases. On the other hand, the relaxation of this SOCP where variables can take rational values (including the $X_{j,l}$'s) can be solved in reasonable time. This solution is not feasible in general but it provides a lower bound on the number of necessary nodes that will be used in order to evaluate the quality of the heuristics we propose.

5.2 Notations

In order to describe the algorithms, we will consider that tasks are sorted by decreasing values of C_j, as usual when designing packing heuristics. Other possible choices would include sorting tasks by decreasing values of $C_j + \rho_j$ and will be discussed in Section 6.2. Let us assume that tasks $T_{j,l}$ have been assigned to node M_k. Then, the load of node M_k will be represented, following the analysis performed in Section 4, by the triplet $S_k = (C_k, x_k, y_k)$, where

$$C_k = \sum_{j,l,\ T_{j,l} \in M_k} C_j, \ x_k = \sum_{j,l,\ T_{j,l} \in M_k} \rho_j \cos(\phi_j), \ y_k = \sum_{j,l,\ T_{j,l} \in M_k} \rho_j \sin(\phi_j).$$

The maximal load of node M_k at any time step t is therefore given by

$$\mathcal{L}(M_k) = C_k + \sqrt{x_k^2 + y_k^2}$$

and becomes $\mathcal{L}(M_k, T_{j,l}) = C_k + C_j + \sqrt{(x_k + \rho_j \cos(\phi_j))^2 + (y_k + \rho_j \sin(\phi_j))^2}$ when one task $T_{j,l}$ of job J_j is added to M_k.

5.3 Heuristics

We propose the following set of heuristics, adapted from classical efficient greedy Bin-Packing algorithms to the case of tasks exhibiting daily patterns.

- First-Fit Decreasing $\mathcal{FFD}$ 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 with the smallest index and such that $\mathcal{L}(M_k, T_{j,l}) \leq C$. If no such node exists, then a new node is added to the system to hold the task.

– 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 to small. In $\mathcal{MM}$, the optimal number of nodes is found using dichotomic search to find the optimal value of M.

– Min-Max-Module $\mathcal{MMM}$ is similar to $\mathcal{MM}$, except that tasks are represented using their maximal demand over time C_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.

6 Experimental evaluation

6.1 Simulated Data

We perform a set of experiments with synthetic data in order to assess the influence of the parameters on the performance of the different heuristics. In all the experiments, we display the ratio between the number of nodes using the heuristics described in Section 5 against the lower bound on the number of necessary nodes described in Section 5.1.

In the following, we set the capacity of the nodes to 20 and we consider the following parameters:

– CPU footprint of the tasks: we consider the case of Big Tasks (where C_j is chosen uniformly at random in $[0, 10]$) and Small Tasks (where C_j is chosen uniformly at random in $[0, 1]$).

– 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 $[0, C_j/2]$).

– Size of the Jobs: we consider the case of Large Jobs (where the number of identical tasks of the job is set to 10) and Small Jobs, which consist in a single task.

In all cases, the phase of each job is chosen uniformly at random in $[0, 2\pi[$. In all the experiments, in order to perform a fair comparison, the expected value of the lower bound is set to 250, so that there are 10 times more tasks in the case of Small Tasks with respect to the case of Big Tasks. 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.

Results for the eight possible combinations (Small or Big tasks / Small or Big amplitude / Small or Big jobs) are displayed in Figure 1.
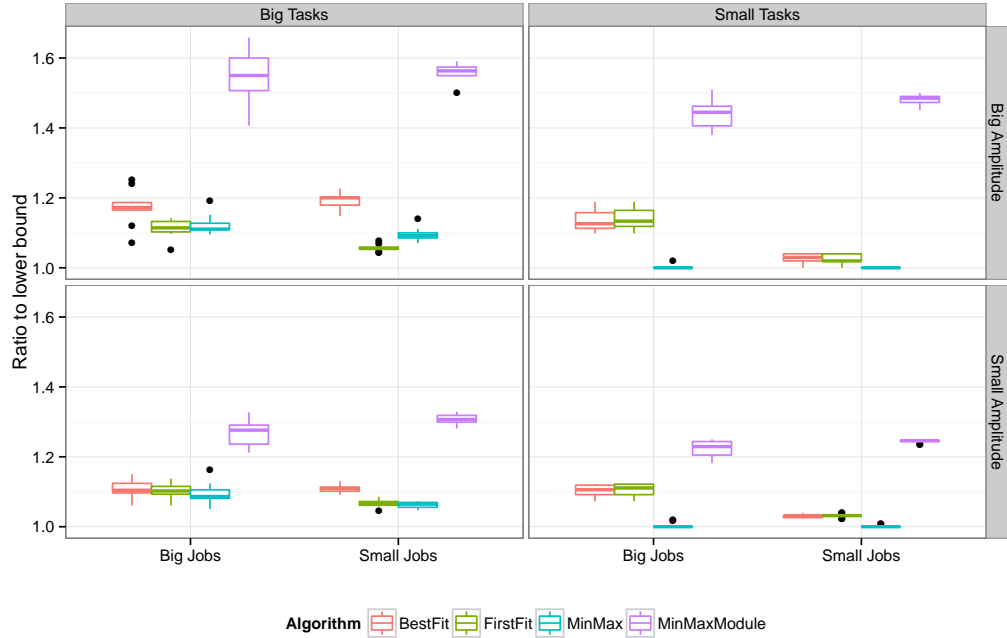

Fig. 1: Performance of the Heuristics on Synthetic Data

The first conclusion that can be drawn is 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.

The second conclusion is that when the tasks are Small, so that each node holds a few tens of tasks, Min-Max $\mathcal{MM}$ performs extremely well and is always at most within 1% of the lower bound. The results of Min-Max $\mathcal{MM}$ slightly degrades when tasks get Big. Indeed, 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. Nevertheless, the number of nodes required by $\mathcal{MM}$ always stays within 20% of the lower bound, and this lower bound is certainly under-estimated, especially in the case of Big Tasks.

In the case of Big Tasks, it happens that First-Fit Decreasing $\mathcal{FFD}$ outperforms Min-Max $\mathcal{MM}$. Indeed, $\mathcal{FFD}$ is an efficient heuristic for classical Bin-Packing problems. On the other hand, it tends to pack together on the same node tasks whose characteristics are close in terms of C_j. In the case of Big Jobs

consisting in several identical tasks, then $\mathcal{FFD}$ packs together tasks that achieve their peak demand at the same time and therefore fails to take full benefit of their periodic behavior.

6.2 Task Ordering

Note that in all the heuristics described in Section 5, tasks are sorted by decreasing values of C_j, whereas their maximal demand is $C_j + \rho_j$. We also tried to sort tasks according to $C_j + \rho_j$ but it degrades the performance of the heuristics. The reason is the following. As observed in Section 5.2, each task can be represented by a triplet (C_j, x_j, y_j) , where $\rho_j = \sqrt{x_j^2 + y_j^2}$ and the state of each node can be represented by a triplet (C_k, x_k, y_k) and the maximal load at any time step is given by $C_k + \sqrt{x_k^2 + y_k^2}$. In practice, the x's and y's can be either positive or negative whereas the C's are always positive. Therefore, the packing heuristics that take periodicity into account tend to annihilate x's and y's and therefore, the amplitude of ρ should not be given as much importance as the amplitude of C when initially sorting the tasks.

In the (most difficult) case of 1000 Big tasks with Big amplitudes, for instance, the number of nodes required by Min-Max $\mathcal{MM}$ heuristic is on average 30% larger than the lower bound when tasks are ordered by decreasing values of $C_j + \rho_j$, whereas the number of nodes required by $\mathcal{MM}$ is on average only 15% larger than the lower bound when tasks are ordered by decreasing values of C_j.

6.3 Jobs and Tasks of Google Trace

As advocated in Section 3, in the trace released by Google [16] and corresponding to one production center, the jobs of the *Normal Production* class that last for the duration of the trace and that exhibit strong daily patterns count for about 50% of the overall load. In this paper, we concentrate on this set of jobs, and we prove that their characteristics make them suitable for the design of efficient resource allocation algorithms, which take into account both their periodic nature and the fact that they do not all reach their peak values at the same time step.

Of course, since this set of jobs accounts for half of the overall demand, it is also crucial to design more dynamic strategies for the rest of the jobs. These jobs typically correspond to the *Gratis (free)* class [13,12] and can be allocated at runtime and then migrated to other nodes when the load of a nodes becomes too high so that the QoS (Quality of Service) of the *Normal Production* class cannot be enforced. Nevertheless, this important problem, addressed in the papers mentioned in Section 2, is out of the scope of this paper.

Following the classification of Section 3, we have extracted 89 jobs corresponding to a total of 22600 tasks. The largest job (in terms of tasks) consists in 1608 tasks. 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. On

the other hand, 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. The results achieved by the different heuristics are displayed in Table 2.

	First-Fit $\mathcal{FFD}$	Best-Fit $\mathcal{BFD}$	Min-Max $\mathcal{MM}$	Min-Max-Module $\mathcal{MMM}$
Number of Nodes	2181	2182	2114	2226

Table 2: Number of nodes required per heuristic.

It can be observed that the results of $\mathcal{MM}$ are extremely good on this actual dataset. Indeed, the number of required machines is only 1.1% higher than the lower bound, whereas $\mathcal{MMM}$, the equivalent heuristic that does not benefit from daily patterns, requires 6.5% more machines than the lower bound. This result proves that there is clear interest to take benefit of daily patterns on an actual dataset.

7   Conclusions

This paper assesses the impact of designing efficient resource allocation algorithms for jobs that exhibit daily periodic sinusoidal patterns. First, we demonstrate that in a trace of a production cluster released by Google, those jobs actually represent a significant part of the workload. Then, we present a novel model of periodic jobs with variable resource demand in shared hosting platforms. We prove that the job aggregation problem, where the objective is to minimize the number of nodes, can be formulated as a SOCP, what enables us to solve it exactly in reasonable time, at least for small instances. We argue that provisioning resources solely based on the maximal demand of tasks, as showed by Min-Max-Module heuristic, results in larger number of nodes. On the other hand, resource provisioning based on an antagonistic job aggregation, as illustrated by the Min-Max heuristic, can yield gains that significantly decrease the number of required nodes. As future work, we plan to extend job aggregation strategies to provide performance guarantees for other resources like memory, disk, network bandwidth, etc. Our future research plans include refining the suggested second order cone program to more efficient mathematical programming relying on *column generation* algorithm. This algorithm is proved to be efficient for solving larger programs as it generates only variables which have the potential to improve the objective function. At last, in order to deal with larger classes of problems, it is crucial to understand how to mix the (close to optimal) strategies used to schedule long-running high priority job classes and the dynamic resource allocation strategies that are used for short and low priority classes.

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, Lionel Eyraud-Dubois, Hejer Rejeb, and Christopher Thraves. Heterogeneous Resource Allocation under Degree Constraints. *IEEE Transactions on Parallel and Distributed Systems*, 2012.

5. 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.

6. 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.

7. 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.

8. Walfredo Cirne and Eitan Frachtenberg. Web-scale job scheduling. In *Job Scheduling Strategies for Parallel Processing*, pages 1–15. Springer, 2013.

9. M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

10. D. Hochbaum. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1997.

11. Hans D Mittelmann. An independent benchmarking of sdp and socp solvers. *Mathematical Programming*, 95(2):407–430, 2003.

12. 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, Carnegie Mellon University, April 2012.

13. 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`.

14. 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.

15. 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.

16. John Wilkes. More Google cluster data. Google research blog, November 2011. Posted at `http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html`.

17. 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.